4-5-2019 3:30 PM

# Local Search Approximation Algorithms for Clustering Problems

Nasim Samei
*The University of Western Ontario*

Supervisor
Professor Roberto Solis-Oba
*The University of Western Ontario*

Graduate Program in Computer Science
A thesis submitted in partial fulfillment of the requirements for the degree in Doctor of Philosophy

# Abstract

In this research we study the use of local search in the design of approximation algorithms for NP-hard optimization problems. For our study we have selected several important and well known clustering problems: $k$-uncapacitated facility location problem, minimum mutliway cut problem, and constrained maximum $k$-cut problem.

We show that by careful use of the local optimality property of the solutions produced by local search algorithms it is possible to bound the ratio between solutions produced by local search approximation algorithms and optimum solutions. This ratio is what is known as the locality gap of the algorithms.

The locality gap of our algorithm for the $k$-uncapacitated facility location problem is $2 + \sqrt{3} + \epsilon$ for any constant $\epsilon > 0$. This matches the approximation ratio of the best known algorithm for the problem, proposed by Zhang but our algorithm is simpler. For the minimum multiway cut problem our algorithm has locality gap 2-2/k, which matches the approximation ratio of the isolation heuristic of Dahlhaus et al; however, our experimental results show that in practice our local search algorithm greatly outperforms the isolation heuristic, and furthermore it has comparable performance as that of the three currently best algorithms for the minimum multiway cut problem (by Calinescu et al, Sharma and Vondrak, and Buchbinder et al). For the constrained maximum $k$-cut problem on hypergraphs we proposed a local search based approximation algorithm with locality gap 1-1/k for a variety of constraints imposed on the $k$-cuts. The locality gap of our algorithm matches the approximation ratio of the best known algorithm for the max k-cut problem on graphs designed by Vazirani, but our algorithm is more general.

**Keywords:** Combinatorial optimization, Approximation algorithms, Local search, Facility location problem, Multiway cut problem, Max k-cut problem

# Co-Authorship Statement

This thesis consists of three research articles that either have already been published or that are currently being considered for publication in specialized journals. The contributions of each one of the authors are listed below. Authors are listed in alphabetical order.

Chapter 2 includes an article titled "Analysis of a local search algorithm for the k-facility location problem" published in the journal RAIRO-Theoretical Informatics and Applications. Nasim Samei provided some research ideas, she contributed to the analysis of the algorithms and wrote the first draft of the paper. Roberto Solis-Oba contributed with research ideas leading to the design of the algorithm and analysis of the algorithms; he also edited the manuscript.

Chapter 3 includes an article titled "A local search algorithm for the multiway cut problem" submitted to Journal of Computer and System Sciences. Andrew Bloch-Hansen implemented the algorithms and performed the experiments; he also wrote the part of the manuscript describing the experimental results. Nasim Samei contributed with research ideas that let to the design of the algorithms and their analysis. She wrote the first draft of the paper. Roberto Solis-Oba contributed with ideas about design and analysis of the algorithm, he also edited the manuscript.

Chapter 4 includes an article titled "A local search algorithm for the constrained max $k$-cut problem on hypergraphs" that is currently under review in Journal of Applied Mathematics and Computing. Nasim Samei contributed with research ideas that let to the design of the algorithms and their analysis. She wrote the first draft of the paper. Roberto Solis-Oba contributed with ideas about design and analysis of the algorithm, he also edited the manuscript.

# Acknowlegements

I would like to express my sincere gratitude to my supervisor, Roberto Solis-Oba, for having strong faith in me. Also, I am very thankful for all his tremendous support in all aspects of my professional development. He is a great mentor and knowledgeable supervisor who provided me with wise advice. This thesis would not be possible without his extraordinary perseverance. I am forever indebted to him.

I am grateful to my examining committee, Yuri Boykov, Hamada Ghenniwa, Daniel Lizotte and Marc Moreno Maza for accepting to be my examiners and their constructive feedback.

I am privileged to have a supportive and understanding family. My greatest thanks to my parents Nastaran and Rasoul for their fantastic support throughout my Ph.D. program especially their great support and patience on my Ph.D. exam date. Also, I am thankful for their great respect for my choices and providing me an environment that I can learn and grow.

# Contents

**Abstract**

**List of Figures** **vi**

**List of Tables** **viii**

**1 An Introduction to Combinatorial Optimization Problems** **1**
1.1 Introduction . . . 1
1.2 Local Search Algorithms . . . 5
1.3 The Complexity of Computing Local Optimum Solutions . . . 10
1.4 Local Search in the Design of Approximation Algorithms . . . 11
1.5 Local Search for Combinatorial Optimization Problems . . . 13
1.5.1 The Multiprocessor Scheduling Problem . . . 13
1.5.2 Computing a Spanning Tree with Many Leaves . . . 14
1.5.3 The $k$-Set Packing Problem . . . 14
1.5.4 The Max $k$-SAT Problem . . . 15
1.5.5 The Traveling Salesman Problem . . . 15
1.5.6 The Quadratic Assignment Problem . . . 16
1.5.7 The $k$-Set Cover Problem . . . 17
1.5.8 The Maximum Constraint Satisfaction Problem . . . 17
1.5.9 The Stable Marriage Problem . . . 18
1.5.10 Placement of Meters in Networks . . . 19
1.6 Our Local Search Algorithms . . . 19
1.6.1 The $k$-Facility Location Problem . . . 19
1.6.2 The Multiway Cut Problem . . . 20
1.6.3 The Constrained Max $k$-cut Problem on Hypergraphs . . . 22
1.7 Organization of the Thesis . . . 25

**Bibliography** **26**

**2 Local Search Algorithm for the $k$-Facility Location Problem** **28**
2.1 Introduction . . . 28
2.1.1 Contributions . . . 29
2.1.2 Organization of the Paper . . . 29
2.2 A Local Search Algorithm with Multiple Swaps . . . 30
2.3 First Bound for the Locality Gap . . . 32

2.3.1 Pairing . . . 33
2.3.2 Analysing the Swaps . . . 35
Multi-Swaps for Sets $A_i$ and $B_i$ where $|A_i| = |B_i| \leq p$ . . . 35
Swaps for Sets $A_i$ and $B_i$ where $|A_i| = |B_i| > p$ . . . 36
Single Swaps for Facilities in Sets $A_r$ and $B_r$ . . . 40
Putting It All Together . . . 40
2.3.3 Special Case When the Ratio of the Biggest Facility Cost to the Smallest Facility Cost Is Less Than $p + 1$ . . . 41
2.4 Tight Example . . . 42
2.5 Scaling the Costs . . . 43
2.5.1 Bounding the Facility Cost . . . 44
2.5.2 Bounding the Service Cost . . . 45

**Bibliography** **48**

**3 A Local Search Algorithm for the Multiway Cut Problem** **50**
3.1 Introduction . . . 50
3.2 The Local Search Algorithm . . . 51
3.3 Finding a minimum cost relabel operation . . . 53
3.4 Algorithm MULTIWAY CUT for the 3-way Cut Problem . . . 56
3.4.1 First bound . . . 58
3.4.2 Second Bound . . . 61
3.4.3 Computing the Approximation Ratio . . . 62
3.5 The Multiway Cut Problem . . . 62
3.5.1 First Bound . . . 63
3.5.2 Second bound . . . 65
3.5.3 Computing the Approximation Ratio . . . 66
3.6 Tight Example . . . 67
3.7 Variations of the Multiway Cut Problem . . . 68
3.7.1 Nodes that Need to be in the Same Partition . . . 68
3.7.2 Nodes that Can Only be in Some Partitions . . . 70
3.8 Experimental Results . . . 71
3.8.1 Input Data . . . 71
3.8.2 Test Cases . . . 72
3.8.3 Results . . . 73
Input Networks . . . 74
Graph Characteristics . . . 75
Running Time . . . 78
Initial Labeling and Value of $\epsilon$ . . . 79
3.8.4 Final Observations . . . 81
3.8.5 Acknowledgements . . . 81

**Bibliography** **82**

**4 Local Search for the Constrained Max $k$-Cut Problem** **84**

4.1 Introduction . . . . . 84
4.2 The Local Search Algorithm . . . . . 87
4.3 Max $k$-Cut, Max Multiway Cut, and Max Steiner $k$-Cut Problems . . . . 88
4.4 Max Capacitated $k$-Cut, Max $k$-Cut with Given Sizes of Parts . . . . . 90
4.5 Directed Max $k$-Cut Problem . . . . . 93

**Bibliography** **99**

**5 Conclusions** **101**
5.1 Main Contributions . . . . . 101
5.2 Challenges of Using Local Search in the Design of Approximation Algorithms 102
5.3 Strengths of Local Search Algorithms . . . . . 103

**Bibliography** **104**

**Curriculum Vitae** **104**

# List of Figures


1.1 An instance of the traveling salesman problem. . . . . 1
1.2 Example of the single swap operation and the corresponding neighborhood function, where $F = \{f_1, f_2, f_3\}$ and $C = \{c_1, c_2, c_3, c_4, c_5\}$. Clients are connected with solid lines to the facilities that service them, and a client is connected to a facility with a dashed line if that facility was servicing the client before performing the swap but it does not service the client after the swap. . . . . 6
1.3 Example of a local optimal solution and a global optimal solution for an instance of the facility location problem. . . . . 7
1.4 Example of a scheduling problem. . . . . 8
1.5 Initial solution for the minimum multiway cut. . . . . 9
1.6 Local optimal solution and global optimal solution for the example of the multiway cut problem in Figure 1.5. . . . . 10
1.7 Instance illustrating a big locality gap for a scheduling problem using the jump operation. . . . . 12
1.8 An example of the 2-opt neighborhood. . . . . 16
1.9 Example of a multi-swap operation, where $S = \{s_1, s_2, s_3\}$, $A = \{s_1, s_2, s_3\}$ and $B = \{f_1, f_2, f_3\}$. . . . . 20
1.10 Instance of the multiway cut problem, where $k = 4$ and $T = \{t_1, t_2, t_3, t_4\}$; one possible solution for this instance of the problem is to divide the vertices into partitions $V_1, V_2, V_3, V_4$ and the cost of this solution is 14. . . 21
1.11 Instance of the labelling version of the multiway cut problem. . . . . 21
1.12 Example of a relabel operation $R\langle A, \alpha, f\rangle$, where $A = \{p, q\}$ and $\alpha = l_3$. . 22
1.13 Instance of the max $k$-cut problem. . . . . 23
1.14 Instance of the max $k$-cut problem on hypergraphs. . . . . 23
1.15 Example of local operations. The dashed edges in Figures $(a_1)$ and $(b_1)$ are the edges that are added to the cost of the solution after performing the corresponding local operation. The dashed edges in Figures $(a_2)$ and $(b_2)$ are the edges that no longer contribute to the cost of the solution after performing the local operation. . . . . 24

2.1 Mapping $\pi$ maps each $o \in S^*$ to its closest facility $\pi(o) \in S$. . . . . 32
2.2 Client $j \in N_S(A_i) \setminus N_S^*(B_i)$ is assigned to $\hat{s} = \pi(\sigma^*(j))$. Note that $\sigma^*(j) \notin B_i$ and $\pi(\sigma^*(j)) \notin A_i$. . . . . 36
2.3 Tight example for $p = 2$ and $q = 5$. . . . . 43
2.4 Tight example for arbitrary $p$ and $q$. . . . . 43

3.1 In graph $G_\alpha$ nodes represented by squares are the nodes in $G$ and nodes labelled with $t_i$ where $i = \{1, 2, 3\}$ are terminals in $G$. Auxiliary nodes are represented by circles. In $G$ labels assigned to the nodes are inside the squares. . . . . 53
3.2 Paths $P$ and $P'$ are shown above in thick and thin solid lines, respectively. 54
3.3 The left figure shows partitions $\hat{A}_1$, $\hat{A}_2$ and $\hat{A}_3$ of the local optimal solution and the right figure shows partitions $A_1^*$, $A_2^*$ and $A_3^*$ of the global optimal solution. In both figures the set of edges that contribute to the 3-way cut connect nodes in different partitions. For example, edge $(u, v)$ in the left figure connects $u \in \hat{A}_1$ to $v \in \hat{A}_2$ so it contributes to the cost of the local optimal solution, and edge $(u, w)$ in the right figure connects $u \in A_1^*$ to $w \in A_2^*$ so it contributes to the cost of the global optimal solution. . . . . 57
3.4 In all the figures, partitions $(\hat{A}_1, \hat{A}_2, \hat{A}_3)$ and $(A_1^*, A_2^*, A_3^*)$ are as in Figure 3.3. Figure (a) represents the set $\Delta_1$, Figure (b) represents set $\Delta_2$, and Figure (c) represents set $\Delta$. . . . . 59
3.5 Node labels are inside the nodes and node names are written beside the nodes. Edge costs are written beside the edges. The terminals are $x_1, x_2, \ldots, x_k$. 68
3.6 An example of the transformation of $G$ into $G'$. Nodes that need to be in the same partition are $s, q, u, p$ so in the transformed graph $G'$ a super node $Q$ is created for these nodes. Also, $cost(t_1, Q) = cost(u, t_1) + cost(t_1, p)$ and $cost(Q, r) = cost(p, r)$. . . . . 69
3.7 Results from the 80 vertex exponential decay graph distribution: Average approximation ratios with 5 terminals (top-left), maximum approximation ratios with 5 terminals (top-right), average approximation ratios with $5n$ edge density (bottom-left), maximum approximation ratios with $5n$ edge density (bottom-right). . . . . 77
3.8 Results from the 160 vertex exponential decay graph distribution with $m = 4n$ and $k = 10$. Approximation ratios are compared against the epsilon value using the first edge capacity scheme (top-left) and the second edge capacity scheme (bottom-left). Running times are compared against the epsilon value using the first edge capacity scheme (top-right) and the second edge capacity scheme (bottom-right). The x-axis shows the value of $\epsilon/k^2$; the percentage of improvement to the previous best solution required to continue the iterations of the algorithm. . . . . 80

4.1 Example of a directed Hypergraph. . . . . 93

# List of Tables

3.1 Weights for the edges in $G_\alpha$. . . . . 54
3.2 Ratios of the solutions computed by approximation algorithms to the optimum, for benchmarks from the DIMACS competitions: Maximum independent set (Brock), maximum clique (Gen, C125), Hamming instances, Keller instances, p-hat instances, and Steiner tree instances (ST). Ratios for the randomly generated instances with simple (SR), linear (GL), and exponential (GE) distributions. . . . . 75
3.3 Average and maximum approximation ratios for several test cases on 80 vertex random graphs with exponential decay distributions. . . . . 76
3.4 Running times using 100 experiments for several test cases from the 80 vertex and 160 vertex exponential decay distributions. . . . . 78

# Chapter 1

# An Introduction to Combinatorial Optimization Problems and Local Search

## 1.1 Introduction

In a combinatorial optimization problem we look for a that either maximizes or minimizes a given objective function. A feasible solution usually requires grouping, ordering or selecting a discrete or finite set of objects that satisfies some given conditions; therefore the solution space (the set of all feasible solutions) for a combinatorial optimization problem is discrete or finite. Usually the *solution spaces* of combinatorial optimization problems are very large and thus using exhaustive search to find optimal solutions, solutions that either maximize or minimize an objective function, is not efficient. Therefore, it is of great interest to design more sophisticated algorithms that can find optimal or near optimal solutions in polynomial time.

A classical example of a combinatorial optimization problem is the traveling salesman problem in which we are given a weighted graph $G = (V, E)$ and the goal is to find a minimum weight *Hamiltonian* cycle. A Hamiltonian cycle is a cycle that includes all the nodes of $G$.

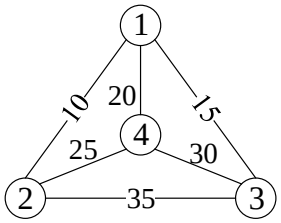


Figure 1.1: An instance of the traveling salesman problem.

Figure 1.1 shows an instance of the traveling salesman problem; the solution space includes all possible Hamiltonian cycles: (2, 1, 4, 3, 2), (2, 1, 3, 4, 2), (2, 4, 1, 3, 2), (2,

4, 3, 1, 2), (2, 3, 1, 4, 2) and (2, 3, 4, 1, 2). The weights of these cycles (the objective function) are: 95, 80, 95, 80, 95 and 95, respectively. A feasible solution is any cycle from the solution space. An optimal solution is any cycle with the smallest weight. In this example cycles (2, 1, 3, 4, 2) and (2, 4, 3, 1, 2) are optimal. Observe that in this example since the size of the solution space is small we can quickly find an optimal solution, however for larger instances the solution space would be huge and it would be very difficult to find optimum solutions.

Combinatorial optimization problems are of great importance because many real life problems can be modeled as combinatorial optimization problems. These problems arise in a wide variety of fields, such as data mining, information retrieval, network routing, image processing, machine learning, artificial intelligence, operation research, and so on.

In addition, combinatorial optimization is of great theoretical importance because it led to advances in other areas such as discrete mathematics, computer science, probability theory, and continuous optimization.

Many important combinatorial optimization problems are NP-hard [2]; this means that there is very strong theoretical evidence suggesting that there are no polynomial time algorithms for solving them. An effective approach for dealing with NP-hard problems is to find solutions that are provably close to the optimal solutions. Algorithms that find these near optimal solutions are called approximation algorithms. To measure the quality of an approximation algorithm we compute its *approximation ratio*. If an algorithm has approximation ratio $\alpha$ then the solutions returned by the algorithm are guaranteed to have values that are within an $\alpha$ factor of the optimal solutions.

We are interested in studying and designing approximation algorithms because, as mentioned earlier, in real life we need to solve NP-hard problems. The analysis of approximation algorithms helps us understand why some approaches work better on some problems and hence it gives us clues on which approaches would be more promising for a new problem. Moreover, approximation algorithms give us a metric on how "hard" different NP-hard problems are.

Due to their importance, there is extensive research in the literature on the design of approximation algorithms. These algorithms have been designed for problems from a large variety of fields and several methodologies have shown to be effective for the design and analysis of approximation algorithms.

We can classify approaches for designing approximation algorithms into two main categories; non-linear programming based approaches and linear programming based approaches. We describe below several non-linear programming-based approaches.

- **Greedy algorithms:** Greedy algorithms usually start with an empty solution and they repeatedly make "greedy" choices that select a "locally" optimal way to enlarge the current solution in the hope that at the end they will find an optimal solution. In these algorithms once a decision is made the decision cannot be modified. Greedy algorithms are usually very fast and they are easy to implement. However, they do not always produce near optimal solutions.

- **Local search algorithms:** A local search algorithm starts with an arbitrary feasible solution, and then it iteratively improves the current solution by selecting a

*neighbor solution* with a better objective function value. The algorithm stops when no further improvement is possible. The neighbors of a given feasible solution are determined by a set of *local operations*. A local operation transforms a given solution $s$ into a new feasible solution $s'$ by usually performing simple changes such as adding, removing or swapping elements of $s$ with some other elements not in $s$ (later we give more examples of local operations). The main reason we are interested in local search is that local search algorithms are conceptually simple and they are usually easy to implement. We believe that local search is a powerful technique that can be used to design efficient approximation algorithms, but so far there are relatively few research works published on the design of local search approximation algorithms. In this thesis we explore the use of local search in the design of algorithms for NP-hard optimization problems with a provable performance guarantee. We will discuss this class of algorithms in more detail in the next section.

- **Dynamic programming:** A dynamic program decomposes a problem into smaller sub-problems that have smaller solution spaces than the original one. The solutions to the smaller sub-problems are combined to construct solutions for bigger sub-problems until we obtain a solution for the entire problem. The main advantages of dynamic programming are: (1) we limit the size of the solution space by breaking down the problem into smaller sub-problems, and (2) in the process of recursively breaking down the problem into simpler sub-problems and considering different ways of combining the solutions of these sub-problems to form a global solution, we might encounter the same sub-problems many times, but they need to be solved only once.

Now, we give a brief introduction to integer and linear programming and linear programming based approaches for the design of approximation algorithms. In an integer program we are given a set of *variables* that can only take integer values and that need to satisfy some *constrains* specified in the form of linear inequalities or linear equations. A *feasible solution* is an assignment of values to the variables that satisfies the constraints. The goal is to find a feasible solution that optimizes a given objective function; the objective function is a linear combination of the variables. An integer minimization program is usually expressed in matrix form as follows,

$$\begin{array}{lrcl} \min & c^T x & & \\ \text{s.t.} & Ax & \leq & b \\ & x & \text{integer} & \end{array} \tag{1.1}$$

where $x$ is a vector containing the variables and $c$ is a vector of cost coefficients; vector $b$ and coefficient matrix $A$ define the constraints of the integer program. Observe that a maximization integer program can be converted to a minimization integer program by simply multiplying each cost coefficient by -1.

Many combinatorial optimization problems can be formulated as integer programs; unfortunately, we do not know polynomial time algorithms that can solve arbitrary integer programs. However, if we relax the integrality constraints such that the variables can take real values, we obtain what is known as a linear program. Linear programs can

be solved in polynomial time, but the solution of a linear program is not guaranteed to assign integer values to the variables; hence solving a linear program does not in general give a feasible solution to a combinatorial optimization problem formulated as an integer program. Each solution of the linear program relaxation of an integer program is called a fractional solution.

There are several linear programming based techniques for designing approximation algorithms; these techniques differ in how fractional solutions are manipulated to produce integer solutions. Some of these techniques are briefly described below.

- **Deterministic rounding:** As mentioned above solving the linear programming relaxation of a combinatorial optimization problem gives a solution that assigns fractional values to the variables, however we are interested in a solution with integer values on the variables. Rounding methods convert a fractional solution to a feasible solution for a combinatorial optimization problem, by rounding up or down the values of the variables to integer values. The rounding must be done in such a way that the value of the rounded solution is close to the value of the fractional one as this ensures that the obtained solution has value close to the optimal.

- **Rounding a dual solution:** In this method a feasible solution to an optimization problem is obtained by rounding the *dual* of the linear program relaxation of an optimization problem. Each linear program $P$ has associated another linear program $P'$, called its dual. Linear program $P$ is called primal. Optimal solutions for primal and dual linear programs have the same value. Furthermore, if $P$ is a maximization linear program $P'$ is a minimization linear program and vice versa. The number of constraints in the primal is equal to the number of variables in the dual and the number of variables in the primal is equal to the number of constraints in the dual. Sometimes, solving the primal linear program is difficult but it is easy to find a solution for the dual, for instance when there are too many variables in the primal. When this happens we solve the dual of the linear program and then we round it to find an approximate solution for the optimization problem.

- **The primal-dual method:** In this method instead of solving the dual of the linear program, we simultaneously build solutions for the linear program and its dual. We start with a feasible but expensive solution for the dual problem and an infeasible solution for the primal. Then we gradually improve the cost of the dual solution while at the same time reducing the number of constraints not satisfied by the primal solution. At the end we get a feasible solution for the primal and a near optimal solution for the dual. By the properties of the primal and dual linear programs, the primal solution also has value close to the optimal.

- **Randomized rounding:** In this method we solve the linear program relaxation of a combinatorial optimization problem and then round it to an integer solution using a randomized algorithm. In the randomized algorithm we assign variables a probability function that determines how likely it is that they get specific integer values. Randomized rounding is a very powerful technique that yields efficient

approximation algorithms for difficult problems that cannot be solved using other techniques.

Even though linear programming based approximation algorithms have beed successfully designed for a large number of NP-hard optimization problems, one of the main drawbacks of these algorithms is their big running times due to the fact that they have to solve linear programs. The focus of this thesis is on the use of local search in the design of approximation algorithms. As we show this technique can be used to design efficient algorithms with good approximation ratios.

## 1.2 Local Search Algorithms

As we mentioned earlier, in combinatorial optimization we look for a solution $S$ from the solution space $\mathcal{A}$, that optimizes an objective function $c : \mathcal{A} \longrightarrow \mathbb{Q}$, where $\mathbb{Q}$ is the set of rational numbers. A local search algorithm is defined by its local operations. A local operation $op(S)$ transforms a solution $S \in \mathcal{A}$ into a new solution $S'$ by for example, adding, removing or exchanging elements of $S$ with elements not in $S$. Local operations are usually simple and they should be able to be performed quickly with algorithms with low running times. A *neighborhood function* $\mathcal{N}$ is defined by the local operations. For each solution $S \in \mathcal{A}$, $\mathcal{N}(S)$ includes all the solutions $S' \in \mathcal{A}$ that can be obtained by performing a local operation on $S$.

The facility location problem is a central problem in combinatorial optimization. In this problem we are given a set $F$ of facilities, a set $C$ of clients, service costs $c$, and opening facility costs $f$. Let $c_{ij}$ be the cost of serving client $i$ by facility $j$ and let $f_j$ be the opening cost for facility $j$. The goal is to select a set $S$ of facilities and to assign clients to facilities in such a way that minimizes the service cost plus the cost of opening the facilities needed to service the clients. Therefore, we want to minimize the following cost function,

$$cost(S) = \Sigma_{i \in S} f_i + \Sigma_{j \in C} c_{j\sigma(j)}, \tag{1.2}$$

where $S$ is a set of facilities and $\sigma(j)$ is the facility in $S$ with the smallest service cost to client $j$. One possible local operation applicable to the facility location problem is the *single swap* operation. A single `swap <f,f'>` operation for a given solution $S$ replaces some facility $f \in S$ with another facility $f' \in F \setminus S$.

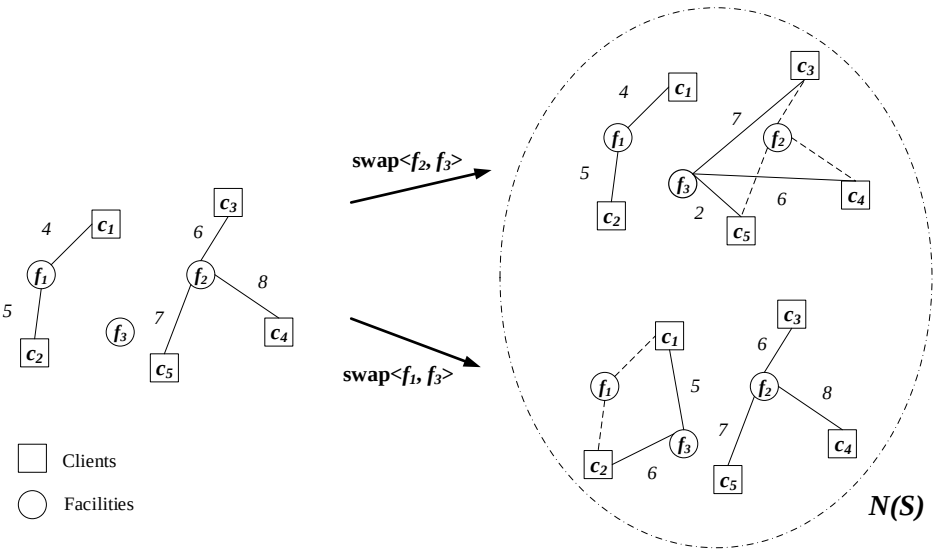


Figure 1.2: Example of the single swap operation and the corresponding neighborhood function, where $F = \{f_1, f_2, f_3\}$ and $C = \{c_1, c_2, c_3, c_4, c_5\}$. Clients are connected with solid lines to the facilities that service them, and a client is connected to a facility with a dashed line if that facility was servicing the client before performing the swap but it does not service the client after the swap.

Consider the example shown in Figure 1.2; assume all the facility opening costs are 0, then the cost of the solution $S = \{f_1, f_2\}$ is equal to the sum of the clients service costs, which is 30. Figure 1.2 shows two neighbouring solutions of $S$ obtained by performing `swap` $< f_2, f_3 >$ and `swap` $< f_1, f_3 >$. By performing `swap` $< f_2, f_3 >$ the cost improves to 24 and by doing `swap` $< f_1, f_3 >$ the cost increases to 32.

There are several variants of local search algorithms. In this thesis we focus on *iterative improvement* local search algorithms, formally described below. Note that a local optimum solution might not be a global optimal solution as we show in the examples presented later. The *locality gap* of a local search algorithm is the largest ratio of the value of a local optimal solution produced by the algorithm to the value of a corresponding global optimal solution. Let $s_{\mathcal{I}}$ be a local optimal solution produced by a local search algorithm for some instance $\mathcal{I}$ of a maximization problem $P$ and let $s^*(\mathcal{I})$ be a global optimal solution for $\mathcal{I}$, then the locality gap of the algorithm is defined as $\min_{\mathcal{I} \in P} \frac{c(s(\mathcal{I}))}{c(s^*(\mathcal{I}))}$, where $c(s(\mathcal{I}))$ is the cost of $s(\mathcal{I})$ and $c(s^*(\mathcal{I}))$ is the cost of $s^*(\mathcal{I})$. Similarly, for minimization problems $P$ the locality gap is defined as $\max_{\mathcal{I} \in P} \frac{c(s(\mathcal{I}))}{c(s^*(\mathcal{I}))}$.

---

Algorithm IterativeImprovement($\mathcal{S}, \mathcal{N}, c$)
**In**: Set $\mathcal{S}$ of feasible solutions, neighborhood function $\mathcal{N}$ and objective function $c$.
**Out**: A local optimum solution $s \in \mathcal{S}$ with respect to $\mathcal{N}$ and $c$.

1. Compute an initial solution $s \in \mathcal{S}$.
2. **while** $\mathcal{N}(s)$ contains a solution with better objective value than $s$ **do** {
3. Choose a solution $s' \in \mathcal{N}(s)$ with better value $c(s')$ than $c(s)$.
4. $s \leftarrow s'$

   }
5. **Output** $s$.

---

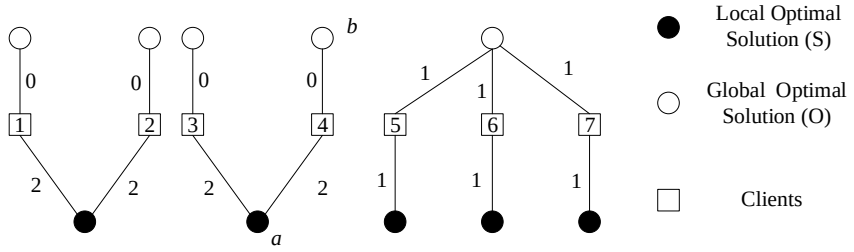


Figure 1.3: Example of a local optimal solution and a global optimal solution for an instance of the facility location problem.

Figure 1.3 shows an instance of the facility location problem with 10 facilities and 7 clients. A local optimal solution $S$ consists of the facilities represented by the black circles and the global optimal solution $O$ contains the facilities represented with white circles; $cost(S)$ is 11 and $cost(O)$ is 3. Observe that $S$ is a local optimal solution with respect to the single swap operation because no single swap operation can improve its cost: For example if we perform `swap<a,b>` then clients 3 and 4 are serviced by facility $b$ at costs 0 and 4 respectively, so the total cost remains unchanged. Observe that $S$ is the worst local optimal solution because no other selection of 5 facilities has a worse cost. Therefore, the local search algorithm with single swap as the local operation has locality gap $\frac{11}{3}$ for this instance of the facility location problem.

We present two more examples to illustrate how a local search algorithm gradually improves an initial solution to obtain a final solution with the property that no further improvement is possible.

We start by considering a scheduling problem in which we are given a set $J$ of $n$ jobs $J = \{j_1, j_2, ..., j_n\}$, a set of $m$ processors $M = \{M_1, M_2, ..., M_m\}$, and processing time $p_i$ for each job $j_i$, $i = 1, 2, ..n$. The goal is to schedule jobs on processors so as to minimize the time needed to complete all the jobs, or the *makespan*. A feasible schedule must satisfy the following conditions: Each processor can process only one job at a time and

the processing of a job cannot be interrupted, i.e. preemptions are not allowed.

Consider an instance of the scheduling problem where $J = \{j_1, j_2, j_3, j_4, j_5, j_6\}$, $M = \{M_1, M_2, M_3\}$ and the processing times are $p_1 = 3$, $p_2 = 2$, $p_3 = 4$, $p_4 = 1$, $p_5 = 2$, and $p_6 = 3$. The solution space includes all possible assignments of jobs to processors. As an example, one possible solution $S$ for this instance is shown in Figure 1.4, in which jobs $j_1, j_2, j_3, j_4$ are processed by $M_1$, $j_5$ is processed by $M_2$, and $j_6$ is processed by $M_3$. The time needed by processor $M_1$ to process jobs $j_1$, $j_2$, $j_3$ and $j_4$ (or the *load* of processor $M_1$) is $3 + 2 + 4 + 1 = 10$. Similarly, the loads of processors $M_2$ and $M_3$ are 2 and 3, respectively. The makespan of the schedule is equal to the maximum load, that in this instance is 10, which is also the time needed to process all the jobs.

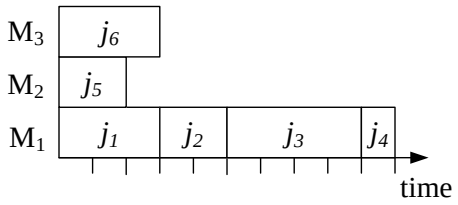


Figure 1.4: Example of a scheduling problem.

In a local search algorithm we try to improve the current solution by performing a set of local operations. For this example we consider a local operation that moves a job from a processor with maximum load to a processor with minimum load; this operation is called the *jump operation*. Therefore, $\mathcal{N}(S)$ includes all the feasible solutions that are obtained from a solution $S$ by performing a jump operation. In the above example one of the neighboring solutions of the solution $S$ shown in the figure can be obtained by moving $j_1$ to processor $M_2$, which results in a reduced makespan. The processing time of $j_1$ is 3, therefore after performing this local operation the loads of the processors $M_1$, $M_2$ and $M_3$ change to $2 + 4 + 1 = 7$, $2 + 3 = 5$, and 3, respectively which decreases the makespan to 7. We can further improve the makespan by moving $j_2$ to $M_3$ and this time the load of all the processors is 5.

After the second jump operation we see that no further improvement is possible, since all the processors have equal load. Therefore, we obtained a *local optimal solution*, where no additional local operations can improve the cost. In this case the local optimal solution is also a global optimal solution because the total load created by all the jobs is $3 + 2 + 4 + 1 + 2 + 3 = 15$ and we have 3 processors, therefore each processor must have load at least 5.

In the multiway cut problem [7] we are given a weighted graph $G = (V, E)$ and a set $T \subseteq V$ of terminals, where $V$ is a set of vertices and $E$ is a set of edges with non-negative weights; the goal is to find a minimum weight set $E' \in E$ whose removal from $G$ separates all the terminals from one another, therefore partitioning $V$ into $|T|$ components each containing one terminal.

Consider the instance of the multiway cut problem shown in Figure 1.5, consisting

of a weighted graph with 6 vertices $V = \{a, b, c, d, e, f\}$ and 3 terminals $T = \{a, b, e\}$. The goal is to partition the graph into 3 components each containing one terminal. The solution space includes all partitions of $V$ that divide $V$ into 3 parts and each part has exactly one terminal. In the figure we indicate vertices that belong to the same partition by assigning them the same label; for the example in Figure 1.5 the labels for the nodes in the three partitions are: $\alpha_1, \alpha_2, \alpha_3$ and the label of each node is indicated inside the node. Hence, the partitions shown in Figure 1.5 are $\{a, d\}$, $\{b, c\}$, $\{e, f\}$. The cost of the solution is 7 because the sum of the weights of the edges in $E' = \{(a, c), (c, d), (b, d), (c, e), (d, f)\}$, whose removal creates this partition, is 7.

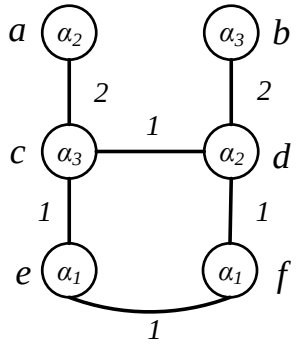


Figure 1.5: Initial solution for the minimum multiway cut.

Consider a local operation that changes the label of a single vertex. The neighborhood defined by this operation for a partition $P$ includes all the partitions that can be obtained from $P$ by changing the label of a single vertex. For the example shown in Figure 1.5 we can change the label of vertex $c$ to $\alpha_1$, and as a result of this operation the cost of the solution decreases by 1. If now we also change the label of vertex $d$ to $\alpha_1$ the cost further deceases by 2. After this second local operation is performed no further operations can improve the cost of the partitioning. Therefore, we obtain a local optimal solution with cost 4. Observe that in this example the local optimal solution is not a global optimal solution since as we can see in Figure 1.6 the cost of a global optimal solution is 3.

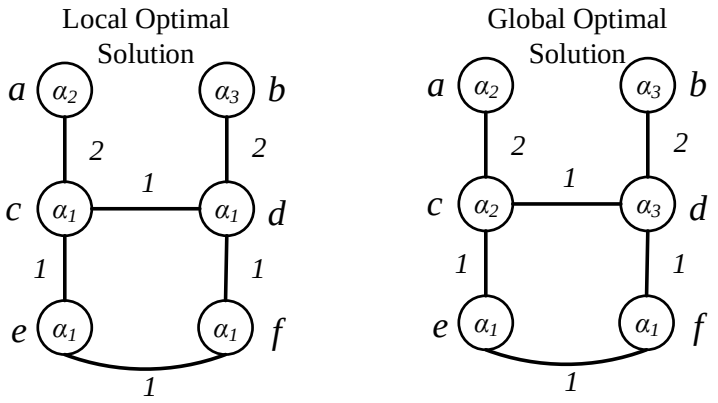


Figure 1.6: Local optimal solution and global optimal solution for the example of the multiway cut problem in Figure 1.5.

## 1.3 The Complexity of Computing Local Optimum Solutions

Local search algorithms in some cases require a long time to compute local optimal solutions due to the fact that local operations could improve the cost of a solution by a very small amount. Let us, for example, consider the IterativeImprovement algorithm presented above. The time complexity of the algorithm depends on the number of iterations of the while loop and on the time it takes to find solution $s'$. Because there is no bound on the amount by which the value of a solution will improve in each iteration, we cannot guarantee a number of iterations that is bounded by a polynomial function of the size of the input.

Research has been conducted to determine the class of optimization problems for which local search algorithms have polynomial running times. Johnson, Papadimitriou, and Yannakakis [16] introduced a class of problems called PLS (polynomial-time local search problems) that admit local search algorithms with polynomial running times.

Orlin, Punnen and Shulz introduced the concept of *$\epsilon$-local optimum solution* [19] that can be used to ensure polynomial running times for some local search algorithms, if a small loss in the quality of the solutions that it produces is tolerated. For a given constant value $\epsilon > 0$, a solution $s$ is $\epsilon$-local optimum (for a minimization optimization problem) with respect to neighborhood function $\mathcal{N}$ if

$$c(s') \geq (1 - \epsilon)c(s) \text{ for each } s' \in \mathcal{N}(s).$$

The notion of $\epsilon$-local optimum solution can be easily extended to maximization problems: $s$ is a $\epsilon$-local optimal solution if $c(s') \leq (1 + \epsilon)c(s)$ for each $s' \in \mathcal{N}(s)$. We can use a technique by Orlin et al. [19] to ensure that the while loop of the IterativeImprovement algorithm performs a polynomial number of iterations: Change the line 3 of the algorithm as follows,

Choose a solution $s' \in \mathcal{N}(s)$, where $c(s') < (1-\epsilon)c(s)$.

Let $s_{ini}$ be the initial solution selected by the IterativeImprovement algorithm and $s_l$ be the $\epsilon$-local optimal solution that it computes. For minimization problems, since each iteration of the algorithm decreases the current solution $s$ by at least $\epsilon \times c(s)$, then the number of iterations of the while loop is $O\left(\frac{\log c(s_{ini}) - \log c(s_l)}{\epsilon}\right)$. Since we are only interested in local search algorithms in which the neighborhood function can be computed in polynomial time, each iteration of the IterativeImprovement algorithm can be performed in polynomial time and so the algorithm runs in polynomial time. We can apply similar arguments to maximization problems as well.

Observe that the quality of an $\epsilon$-local optimum solution is lower than that of a local optimum solution because for a minimization problem instead of each solution $s' \in \mathcal{N}(s)$ satisfying the condition $c(s') \geq c(s)$, where $s$ is the local optimum solution, the $\epsilon$-local optimum solution $s$ satisfies the weaker condition $\frac{1}{1-\epsilon}c(s') \geq c(s)$; therefore an $\alpha$-approximation local search algorithm yields an $\epsilon$-local search algorithm that produces solutions of value within a factor $\frac{1}{1-\epsilon}\alpha$ of the global optimal solutions. Similarly, for maximization problems an $\alpha$-approximation algorithm can be converted into an $\epsilon$-local search algorithm that gives solution of value within a factor $(1+\epsilon)\alpha$ of the global optimal solutions. In Chapter 2 we show how to use this notion of $\epsilon$-local optimum to design a polynomial time local search algorithm for the $k$-facility location problem[1].

## 1.4 Local Search in the Design of Approximation Algorithms

In spite of the conceptual simplicity of local search algorithms, they have not been used extensively for designing approximation algorithms. The two most significant reasons that make it difficult to design local search approximation algorithms for optimization problems are: First, computing the locality gap of a local search algorithm is complicated as we will see in Chapters 2, 3 and 4, and second, in some cases the locality gap of an algorithm is large and finding alternative local operations that would lead to efficient local search algorithms is difficult.

To clarify the second reason let us consider of the scheduling problem presented in Section 1.2. Consider an instance of this problem when the number of jobs $n$ is an even multiple of the number $m$ of processors, or in other words, $n = km$ for some even integer $k > 0$. Assume all jobs have unit processing time. If all the jobs are assigned evenly to only two processors, $M_1$ and $M_2$, as shown in Figure 1.7 then by using the jump operation, as discussed in Section 1.2, the makespan cannot be improved; therefore, the solution given in Figure 1.7 is locally optimal with makespan $\frac{km}{2}$. A global optimal solution for this example is obtained by distributing jobs evenly among all the processors and this achieves a makespan of $k$. Therefore, the IterativeImprovement algorithm with the jump operation has a locality gap of at least $\frac{km/2}{k} = \frac{m}{2}$, which is very large when $m$ is big.

[1]Variant of the facility location problem in which we can open at most $k$ facilities.

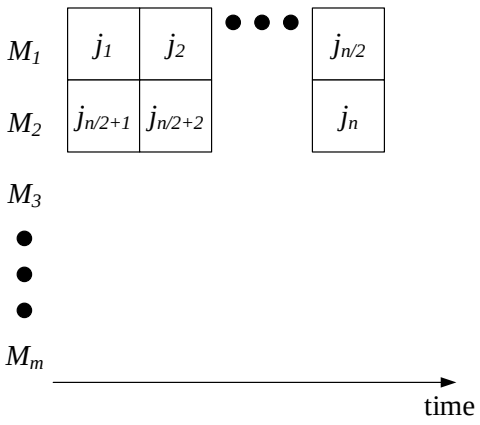


Figure 1.7: Instance illustrating a big locality gap for a scheduling problem using the jump operation.

The problem with some local search algorithms is that they might get "trapped" in a locally optimal solution that is far away from a global optimal solution. The above example clearly illustrates how this is possible. To address this problem, different classes of local search algorithms have been proposed such as: variable-depth search, tabu search, simulated annealing and genetic algorithms [4]. However, in this work we only consider iterative improvement algorithms. The other local search techniques are complicated and it is very difficult to compute their locality gaps. For the rest of this thesis "local search" refers to iterative improvement.

Sometimes by using a different local operation we can improve the locality gap of a local search algorithm. For example, if we change the jump operation in the previous example in such a way that instead of moving only one job at a time we move up to $n$ jobs simultaneously then we can guarantee a locality gap of 1, or in other words this local search algorithm would be an optimal algorithm. We call a local search algorithms with locality gap 1 *exact*.

However, by changing the local operation the size of the neighborhood $|\mathcal{N}(s)|$ for a given solution $s$ can grow exponentially; hence, as a result of changing the local operation the time complexity of a local search algorithm can become exponential as it would be the case with the above jump operation that can move $n$ job simultaneously. In the design of efficient local search algorithms besides achieving a small locality gap we need to achieve a reasonable time complexity.

In this thesis we present local search approximation algorithms for three important and well known combinatorial optimization problems: $k$-uncapacitated facility location, minimum multiway cut and maximum $k$-cut. As we show in Chapters 2, 3 and 4, to compute the locality gaps of our algorithms we use the *local optimal property*: The cost of a local optimal solution is no worse than the cost of all its neighboring solutions. We show how to use this property to obtain a set of inequalities that relate the cost of parts of a local optimum solution with the cost of parts of a global optimum solution. We need to select these inequalities carefully so that when combining them we get an estimate of the cost of the local optimal solution in terms of the cost of a global optimal solution.

## 1.5 Local Search Approximation Algorithms for Combinatorial Optimization Problems

In this section we give an overview of research that has been conducted on the design of local search approximation algorithms.

### 1.5.1 The Multiprocessor Scheduling Problem

As mentioned in Section 1.4 a problem with local search algorithms is that they might get trapped in a locally optimal solution that is far away from the global optimal one. Recall the example in Section 1.4 where two processors have the same maximum load and all other processor have load zero, and all other processor have lead zero and by performing a jump operation the makespan does not decrease.

To deal with this issue we can force the iterative improvement algorithm to continue performing jump operations on the jobs scheduled on processors with the maximum load. To achieve this, we define a new objective function $c'$ that assigns to every solution $s$ a pair $(c(s), \ell(s))$, where $c(s)$ is the makespan of $s$ and $\ell(s)$ is the number of processors with load $c(s)$. Given two solutions $s$ and $s'$, $c'(s) < c'(s')$ if $c(s) < c(s')$ or $c(s) = c(s')$ and $\ell(s) < \ell(s')$.

The neighborhood $\mathcal{N}(s)$ of $s$ has size $O(mn)$ because for all the $n$ jobs there are $m-1$ processors where a job can be moved (all processors except the processor on which the job is currently scheduled). Brucker et al. [3] show that the time complexity of the IterativeImprovement algorithm with the jump operation and cost function $c'$ is $O(mn^3)$ and that its approximation ratio is $2 - \frac{1}{m}$.

**THEOREM 1** *The IterativeImprovement algorithm with the jump neighborhood function and cost function $c'$ has locality gap $2 - \frac{1}{m}$.*

**Proof** The proof that we give here for this theorem is very similar to the proof by Graham for the approximation ratio of his List scheduling algorithm [12]. Let $s$ be the local optimal solution computed by the IterativeImprovement algorithm and let $s^*$ be a global optimal solution. Let $M_i$ be a processor with maximum load, let $j_r$ be the last job processed by $M_i$ and let $t$ be the time when $M_i$ starts to process $j_r$; therefore, $c(s) = t + p_r$. Observe that since $s$ is a locally optimal solution then no processor can have load less than $t$. To see this let us consider that there is a processor $M_k$ with load $l_k < t$; then by moving $j_r$ to $M_k$ we either obtain a solution with smaller makespan or we reduce the number of processors with maximum load, contradicting the locality optimal condition. Therefore,

$$t \leq \frac{1}{m}\left(\sum_{j_i \in J} p_i - p_r\right) \leq c(s^*) - \frac{1}{m}p_r, \tag{1.3}$$

as no solution can have makespan smaller than $\frac{1}{m}\sum_{j_i \in J} p_i$. Therefore,

$$\begin{aligned} c(s) = t + p_r &\leq c(s^*) - \frac{1}{m}p_r + p_r \\ &= c(s^*) + (1 - \frac{1}{m})p_r \\ &\leq c(s^*) + (1 - \frac{1}{m})c(s^*) \text{ as } p_r \leq c(s^*) \\ &= c(s^*)(2 - \frac{1}{m}). \end{aligned}$$

### 1.5.2 Computing a Spanning Tree with Many Leaves

Given a graph $G = (V, E)$, a *spanning tree* of $G$ is a subgraph of $G$ that includes all vertices in $V$ and it is a *tree*, i.e. it does not have any cycles. In a tree $T$ we call all vertices with degree one *leaves*. We describe a local search approximation algorithm by Ravi and Lu [18] for the problem of finding a spanning tree with maximum number of leaves for a given graph $G$. The solution space of this problem includes all the spanning trees of the input graph $G$. The objective function $c(t)$ is the number of leaves in spanning tree $t$.

For a spanning tree $t$ the *exchange neighborhood* of $t$ includes all the spanning trees that differ from $t$ by a single edge. Observe that the number of edges in a spanning tree $t$ is $n - 1$, where $n$ is the number of vertices of $G$. The exchange neighborhood of $t$ is then defined as follows:

$$\mathcal{N}(t) = \{t' \mid t' \text{ is a spanning tree of G and } |t \cap t'| = n - 2\}.$$

For a given spanning tree $t$ the local exchange operation replaces an edge of $t$ with an edge in $G - t$ that results in a new spanning tree $t'$. Ravi and Lu showed that $|\mathcal{N}(t)|$ has size at most $O(mn)$ and that the time complexity of their algorithm is $O(mn^2)$.

We do not show how to compute the locality gap for this algorithm as this analysis is much more complicated than for the above scheduling problem; the reason is that for the scheduling problem we could find a bound for the value of the global optimal solution that we could easily relate to the value of the local optimum solution. However, for the problem of finding a spanning tree with maximum number of leaves there does not seem to be a bound for the global optimal solution that can be easily related to the local optimal solution.

Ravi and Lu [18] showed that their algorithm has a locality gap of 10. In addition, they showed that by using a different local operation that allows the simultaneous exchange of two tree edges with two non-tree edges the locality gap can be improved to 3.

### 1.5.3 The $k$-Set Packing Problem

In the *$k$-set packing problem* we are given a finite set $E$ of $n$ elements and a collection $F$ of $m$ subsets of $E$, where each subset in $F$ has at most $k$ elements. The goal is to find a maximum cardinality sub-collection of $F$ formed by pairwise disjoint sets. In the

weighted version of this problem, called *W-k-set packing*, each set is assigned a cost and the goal is to find a maximum cost collection of pairwise disjoint sets. Both problems are NP-hard [11].

Perhaps the simplest approach to solve the $W$-$k$-set packing problem is a greedy approach that starts with an initially empty collection $S$, and then it adds to $S$ a maximum cost set $B$ in $F$ and removes from $F$ all the sets intersecting $B$. This process is repeated until the collection $F$ becomes empty.

We show that the above greedy algorithm has approximation ratio $k$. Let $S = \{S_1, S_2, ..., S_r\}$ be the collection selected by the greedy algorithm and let $S^* = \{S_1^*, S_2^*, ..., S_p^*\}$ be an optimum solution. Sets in $S$ are indexed in the order in which they were selected by the greedy algorithm. Then, for set $S_1$ there are at most $k$ sets in $S^*$ that have a non-empty intersection with $S_1$ because all the sets in $S^*$ are disjoint and have at most $k$ elements. Also, the cost of each one of these sets is no greater than that of $S_1$. We remove these sets from $S^*$. For set $S_2$ there are at most $k$ sets in the remaining sets of $S^*$ that intersect it and their costs are no greater than the cost of $S_2$ because otherwise they would have been selected as set $S_2$ by the greedy algorithm. Continuing in the same manner we conclude that for each set $t$ in $S$ there are at most $k$ sets in $S^*$ that intersect it and have costs no greater than the costs of $t$. Therefore, the optimum solution has cost at most $k$ times greater than the cost of $S$.

Chandra and Halldörsson [5] combined this greedy algorithm and a local search approach to design an approximation algorithm for the $W$-$k$-set packing problem with locality gap $(\frac{n}{n-1})(\frac{4k+2}{5})$. Their algorithm is a modification of the IterativeImprovement algorithm where they used the above greedy algorithm to find the initial solution.

### 1.5.4 The Max $k$-SAT Problem

Let $B = \{x_1, x_2, ..., x_n\}$ be a set of $n$ Boolean variables. A literal is defined to be either a variable $x_i$ or its negation $\bar{x_i}$, $i = 1, .., n$. In the MAX $k$-SAT problem we are given a set of $m$ boolean clauses $C = \{C_1, C_2, ..., C_m\}$ where, each clause is a disjunction of exactly $k$ literals. The goal is to assign values to the variables so as to maximize the number of clauses that are satisfied. A clause is satisfied if its value is true.

Let $s = (s_1, ..., s_n)$ be a vector that represent a solution to the MAX $k$-SAT problem, i.e $s_i$ is the value for variable $x_i$. The *flip neighborhood* of $s$, $\mathcal{N}(s)$, consist of all the solutions that can be obtained by changing only one value in vector $s$, i.e, by changing one value from true to false or vice versa.

Hansen and Jaumard [14] designed a local search algorithm for the MAX $k$-SAT problem with the flip neighborhood function and show that it has locality gap $\frac{k+1}{k}$.

### 1.5.5 The Traveling Salesman Problem

In the famous traveling salesman problem we are given a weighted graph $G = (V, E)$ and the goal is to find a *Hamiltonian cycle*, a path in $G$ with the same start and end vertex that visits all the vertices in $V$ once, of minimum weight. For this problem we define the cost function as the sum of the weights of the edges in a cycle; therefore, in this problem we are looking for a minimum cost Hamiltonian cycle.

The optimum solution to the traveling salesman problem cannot be approximated within a constant factor in polynomial time unless $P = NP$ [20]. However, if we assume that the weights of the edges satisfy the *triangle inequality*, i.e. for any three edges $e_1$, $e_2$ and $e_3$ forming a cycle of length 3 the sum of the weights of any two edges is greater than or equal to the weight of the remaining edge, then it is possible to design approximation algorithms with constant approximation ratio. This problem is called the metric traveling salesman problem.

Given a cycle $T$, a cycle $T'$ is a 2-opt neighbor of $T$ if it can be obtained from $T$ by removing two edges from it and adding two new edges (see Figure 1.8).

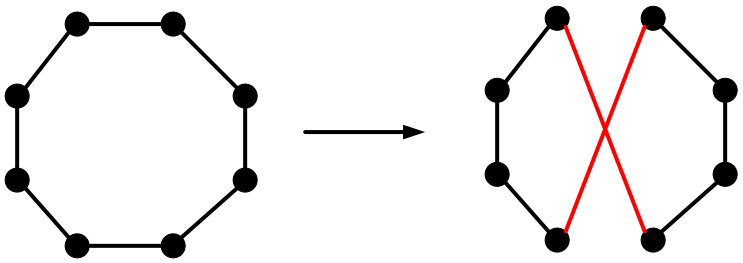

Figure 1.8: An example of the 2-opt neighborhood.

Chandra, Karloff and Tovey [6] designed a local search algorithm with the 2-opt neighborhood for the metric traveling salesman problem and proved that it has locality gap $4\sqrt{n}$, where $n$ is the number of the vertices in $G$.

The related *maximum weight Hamiltonian circuit* is similar to the traveling salesman problem, however instead of finding a minimum weight cycle the goal now is to find a maximum weight cycle. Fisher, Nemhauser and Wolsey [9] showed that a local search algorithm with the 2-opt neighborhood has locality gap 2 for the maximum weight Hamiltonian circuit problem.

### 1.5.6 The Quadratic Assignment Problem

In the quadratic assignment problem, denoted by QAP, we are given a set of facilities $F_s = \{f_1, f_2, ..., f_n\}$, a set of locations $L = \{l_1, 1_2, ..., l_n\}$, a flow matrix $F = (f_{ij})$, where $f_{ij}$ is the flow of material from facility $i$ to facility $j$, and a distance matrix $D = (d_{ij})$, where $d_{kl}$ is the distance from location $k$ to location $l$. The cost of assigning facilities $i$ and $j$ to locations $k$ and $l$ is defined to be $f_{ij}d_{kl}$. The goal is to find an assignment of facilities to locations, or in other words a permutation $\pi$ that assigns a facility $i$ to a location $\pi(i)$, that minimizes the total cost of the assignment, given by the following sum

$$\sum_{1 \le i \le n} \sum_{i+1 \le k \le n} f_{ik} d_{\pi(i)\pi(k)}. \tag{1.4}$$

Angel and Zissimopoulos [1] used a local search algorithm with the following 2-exchange neighborhood and proved that $C_{loc} \le \frac{n}{2} C_{AV}$, where $C_{loc}$ is the cost of the

solution obtained by their local search algorithm and $C_{AV}$ is the average cost over all possible solutions for any instance of $QAP$.

For a permutation $\pi = (\pi(1), \pi(2), ..., \pi(i), ..., \pi(j), ..., \pi(n))$ the 2-exchange neighborhood of $\pi$ includes all the $\frac{n(n-1)}{2}$ permutations of the form $(\pi(1), \pi(2), ..., \pi(j), ..., \pi(i), ..., \pi(n))$ for $1 \leq i < j \leq n$ obtained by performing a swap of $\pi(i)$ and $\pi(j)$ in $\pi$.

### 1.5.7 The $k$-Set Cover Problem

In the $k$-set cover problem we are given a set $U$ and a collection $C$ of subsets of $U$ each of size at most $k$. The goal is to find a minimum size sub-collection of $C$ whose union is $U$. A set of size $k$ is called a $k$-set. Without loss of generality we can assume that $C$ is closed under subsets.

The $(s,t)$-neighborhood for a given solution $S$ for this problem is determined by two values $s, t > 0$ and includes all the solution $S'$ obtained from $S$ in two steps: First insert up to $s$ 3-sets into $S$ and delete up to $t$ 3-sets from $S$. Since not all elements of $U$ might be covered after performing this first step, a second step is performed that optimally selects 2-sets and 1-sets to cover all the missing elements from $U$. The process of optimally selecting 2-sets and 1-sets for the second step can be done by modeling this problem as a maximum matching problem as follows. Construct a graph $G = (V, E)$, where $V$ is the set of uncovered elements in $U$ and $E$ is determined by the 2-sets in $C$, i.e. if there is a 2-set $A = \{a, b\}$ in $C$ then add an edge to $E$ between the vertices corresponding to $a$ and $b$. The optimal 2-sets are selected by finding a maximum matching in $G$ and the 1-sets are the remaining vertices that are not covered by the maximum matching (remember that $C$ is closed under subsets).

The quality of a solution $S$ is determined by the number of sets in $S$ (a solution with fewer sets is better). An $(s,t)$-improvement is a move from a solution $S$ to a $(s,t)$-neighboring solution $S'$ with better quality. Duh and Fürer [8] designed a local-search algorithm that uses (2,1)-improvements. They showed that their algorithm for the 3-set cover problem has a locality gap of $\frac{4}{3}$ and that this bound is tight.

### 1.5.8 The Maximum Constraint Satisfaction Problem

To introduce the maximum constraint satisfaction problem, denoted MAX-CSP, first we define the notion of *constraint satisfaction*. For a given graph $G = (V, E)$ an *assignment* $\pi$ is a function that assigns to each vertex a value from the set $D = \{1, ..., k\}$. A *constraint* $R(u, v)$ between vertices $u$ and $v$ defines a set of pairs of values that can be assigned to $u$ and $v$. A constraint $R(u, v)$ is *satisfied* by assignment $\pi$ if and only if $(\pi(u), \pi(v))$ is in $R(u, v)$.

In MAX-CSP we are given a graph $G = (V, E)$ with a constraint $R(u, v)$ associated with each edge $(u, v) \in E$, and a positive integer $k$; the goal is to find an assignment $\pi : V \to \{1, 2, ..., k\}$ such that the number of satisfied constraints is maximized.

A constraint $R(u, v)$ is *r-consistent* for $1 \leq r \leq k$ if and only if, for every value $x$, $1 \leq x \leq k$, there are at least $r$ values for $y$ and $z$ such that $(x, y) \in R(u, v)$ and $(z, x) \in R(u, v)$. An instance of MAX-CSP is called $r$-consistent, and it is denoted as MAX-CSP$(k, r)$, if and only if all its constraints are $r$-consistent.

Halldórsson [13] defined a neighborhood of an assignment $\pi$ formed by all the assignments obtained from $\pi$ by changing the value of only one vertex. He showed that a local search iterative improvement algorithm with the above neighborhood function has approximation ratio $\frac{r}{k}$ for MAX-CSP$(k, r)$.

### 1.5.9 The Stable Marriage Problem

In the stable marriage problem (SM) we are given two sets, usually called men and women, each of size $n$. Elements of each set rank the elements of the other set. The goal is to find a matching between the two sets, i.e match each man with a woman, such that there are no two elements of opposite sex that prefer to be matched together than with their current matchings. A matching with this property is called *stable*. There is a variant of the SM problem called *SMTI* that allows *ties* and *incompleteness*. Ties allow indifference in the ranked list of each element and incompleteness allows each element to accept only certain subset of elements as partners.

A *SMTI marriage* $M$ is a one-to-one matching between the two sets such that all the matched elements accept each other. A woman $w$ is matched with a man $m$ in $M$ if $M(m) = w$ and $M(w) = m$. If there is no match for a person $p$ then we call $p$ single. The *marriage size*, denoted $|M|$, is defined as the number of men (women) who have a match in $M$. A *blocking pair* $(m, w)$ in marriage $M$ is a pair such that $m$ and $w$ accept each other and $m$ is either single in $M$ or strictly prefers $w$ than its current marriage $M(m)$ and vice versa, i.e. $w$ is either single in $M$ or strictly prefers $m$ than its current marriage $M(w)$. A marriage $M$ is called a *stable marriage* if and only if it has no blocking pairs.

The problem of finding the largest stable marriage is denoted as *MAX SMTI* and we now describe a local search algorithm for it. The neighborhood $N(M)$ of marriage $M$ includes all the marriages that are obtained from $M$ by finding a *good* partner for one of the single men in $M$. A good partner is a match such that the new couple does not create a blocking pair.

Iwama and et al. [15] designed a local search algorithm for the MAX SMTI problem with the above neighborhood and obtained a locality gap of 1.875. Their local search algorithm first selects an arbitrary matching $M$ using the *Gale-Shapley* algorithm and then it repeatedly selects an appropriate neighbor from $N(M)$ for the current matching $M$.

The Gale-Shapley algorithm works as follows: At the beginning of the algorithm all men and women are single. At each iteration of the algorithm a single man selects a highest ranked available woman in his preference list and makes a proposal to her. If a woman receiving a proposal is single she accepts it and becomes engaged, otherwise if she was engaged to another man then she compares him with the new man proposing her and accepts the more preferable one. A man who is refused by a woman becomes (or remains) single. This process is finished when there is no single man remaining with a possible proposal option. Gale and Shapley [10] showed that the matching obtained by the above algorithm is stable.

### 1.5.10 Placement of Meters in Networks

To measure commodity flow on a flow network one could measure the flow on all the edges of the network; however, by using the flow conservation property we can achieve the same goal by only measuring the flow on the edges of any *feedback edge set* and then infering the flow on the remaining edges. A *feedback edge set* (FES) of a graph $G = (V, E)$ is a set $E' \subseteq E$ whose removal converts a graph into an acyclic graph. The number of flow meters needed to obtain the flow on all the edges of a network is then equal to the number of edges in a FES. There is another option for measuring the flow on edges through the use of pressure meters. Pressure meters are placed on some nodes of the network and the flow on an edge is computed by measuring the flow pressure on the nodes incident to the edge. We could place the pressure meters on all the nodes of the network, however a more efficient solution is to put the pressure meters only on the nodes incident on an FES. Therefore, to minimize the number of meters needed, we wish to find a FES with the minimum number of nodes incident on it.

The problem of finding the minimum number of pressure meters on graphs is equivalent to the *minimum vertex feedback edge set* (VFES) problem, where given a graph $G = (V, E)$ the goal is to find a FES with minimum number of vertices incident on it.

Khuller and et al. [17] showed that any minimal FES obtained by repeatedly deleting an edge from each cycle in the given graph has approximation ratio 3 and they also designed a local search algorithm with locality gap $2+\frac{1}{k}$ using the following neighborhood function: for a given constant $k$ the neighbors of a given FES $f$ are all the feedback edge sets that differ from $f$ by at most $k-1$ edges.

## 1.6 Our Local Search Algorithms

We introduce in this section three combinatorial optimization problems that we have studied: $k$-facility location, multiway cut, and max $k$-cut. We also describe the local operations and neighborhood functions that we used in the local search algorithms that we designed for these problems.

### 1.6.1 The $k$-Facility Location Problem

In the $k$-facility location problem ($k$-UFL) we are given a set $F$ of facilities, a set $C$ of clients, service costs $c$, facility costs $f$, and an integer $k$ bounding the maximum number of facilities that can be selected for servicing the clients. The goal is to select a set $S$ of at most $k$ facilities to service the clients that minimizes the facility cost plus the service cost. Therefore, we want to minimize the following cost function,

$$cost(S) = \Sigma_{i \in S} f_i + \Sigma_{j \in C} c_{j\sigma(j)}, \tag{1.5}$$

where $S$ is set of at most $k$ facilities, $\sigma(j)$ is the facility in $S$ with the smallest service cost to client $j$, $f_i$ is the cost of facility $i$ and $c_{j\sigma(j)}$ is the cost of servicing client $j$. In Chapter 2 we present a local search algorithm for the *metric* version of the $k$-facility

location problem. In the metric version of the problem the service costs satisfy the triangle inequality.

For example, consider the instance of the $k$-UFL problem shown in Figure 1.9. The facility costs are inside the nodes and the service costs are beside the edges. Let $k = 3$ and $S = \{s_1, s_2, s_3\}$. The cost of solution $S$ is 22 because the facility cost is $4+4+4 = 12$ and the service cost is $2 + 2 + 2 + 2 + 2 = 10$.

We designed a local search algorithm for the $k$-UFL problem that uses a *multi-swap* operation as local operation: Given a solution $S$ for the problem, in a multi-swap operation we replace all the facilities in some set $A \subseteq S$ with the facilities in another set $B \subseteq F \setminus S$. Formally,

`swap <A,B> :=` $(S \setminus A) \cup B$, where $A \subseteq S$ and $B \subseteq F \setminus S$.

Consider the same instance of the $k$-UFL problem in Figure 1.9 and the above solution $S$. Let $A = \{s_1, s_2, s_3\}$ and $B = \{f_1, f_2, f_3\}$. By performing `swap <A,B>` the cost of the solution decreases from 22 to 8.

Using this local multi-swap operation the neighbors of a given solution $S$ are all the solutions of the form $(S \setminus A) \cup B$, where $A \subseteq S$ and $B \subseteq F \setminus S$.

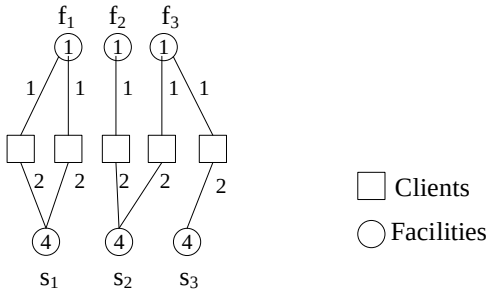


Figure 1.9: Example of a multi-swap operation, where $S = \{s_1, s_2, s_3\}$, $A = \{s_1, s_2, s_3\}$ and $B = \{f_1, f_2, f_3\}$.

### 1.6.2 The Multiway Cut Problem

The multiway cut problem is another well-known combinatorial optimization problem, in which we are given a weighted graph $G = (V, E)$, an integer $k$, and a set of terminals $T = \{t_1, t_2, \ldots, t_k\} \subseteq V$. The goal is to divide the set $V$ of vertices into $k$ disjoint partitions in such a way that each partition has exactly one terminal and the sum of the weights of the edges with endpoints in two different partitions is minimized.

An instance of the multiway cut problem is shown in Figure 1.10, where $k = 4$ and $T = \{t_1, t_2, t_3, t_4\}$. One possible solution for this instance of the problem is to divide the vertices into the partitions $V_1$, $V_2$, $V_3$, $V_4$ shown in the figure. Let us call this solution $P$. Let $R$ be the set of edges crossing the partitions or, in other words, the edges that have their endpoints in two different partitions. These edges are drawn as dashed lines in Figure 1.10. The total weight of the edges in $R$ is 14, therefore the cost of the solution $P$ is 14.

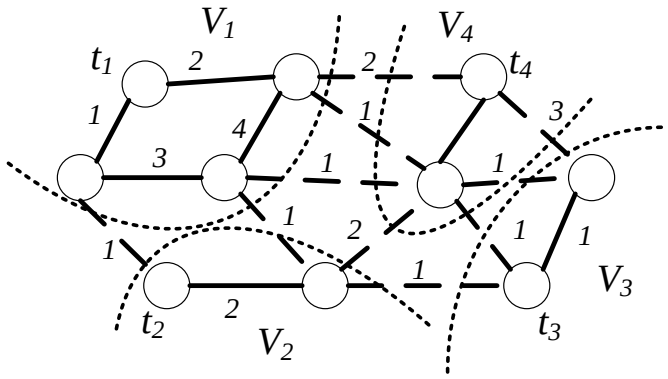


Figure 1.10: Instance of the multiway cut problem, where $k = 4$ and $T = \{t_1, t_2, t_3, t_4\}$; one possible solution for this instance of the problem is to divide the vertices into partitions $V_1, V_2, V_3, V_4$ and the cost of this solution is 14.

We designed a local search algorithm for the multiway cut problem that uses the *relabel* operation as local operation. The relabel operation is easier to explain in the context of labeling problems; therefore first we formulate the multiway cut problem as a labelling problem. In the labelling version of the multiway cut problem we are given a weighted graph $G = (V, E)$, a set of terminals $T = \{t_1, t_2, \ldots, t_k\} \subseteq V$, and a set of labels $L = \{l_1, l_2, \ldots, l_k\}$; the goal is to assign to each node a label from $L$ in such a way that the terminals have different labels and the sum of the weights of the edges that have their endpoints labelled with two different labels is minimized. An example of the labelling version of the multiway cut problem is shown in Figure 1.11, where $k = 4$ and $L = \{l_1, l_2, l_3, l_4\}$. Figure 1.11 illustrates one possible labelling $f$ that defines partitioning $P = \{V_1, V_2, V_3, V_4\}$ where all nodes in $V_i$ are labelled $l_i$, for all $i = \{1, 2, 3, 4\}$. Since partitioning $P$ is the same as that shown in Figure 1.10 the cost of $P$ is 14.

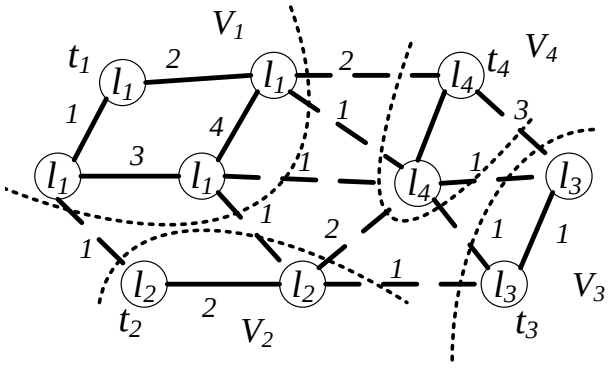


Figure 1.11: Instance of the labelling version of the multiway cut problem.

It is not hard to see that there is a one-to-one correspondence between instances of the multiway cut problem and instances of the above labelling problem. Therefore, a solution $P = \{V_1, V_2, \ldots, V_k\}$ for the multiway cut problem determines a labeling $f$

where all the nodes in partition $V_i$ are assigned label $l_i$, and viceversa, a solution $f$ to the labeling problem determines a partitioning $P$ in which all the nodes labeled $l_i$ form a partition $V_i$ (see Figures 1.10 and 1.11).

Given a labelling $f$ for the vertices of a graph $G = (V, E)$, a relabel operation changes the labels of the nodes in some set $A \subseteq V \setminus T$ to a given label $\alpha$ leaving the labels of all other nodes unchanged. Formally, given a labelling $f$ for the vertices, a set $A$ of vertices, and a label $\alpha$, relabel operation $R\langle A, \alpha, f\rangle$ is defined as follows,

$$R\langle A, \alpha, f\rangle := f(u) = \alpha, \forall u \in A.$$

As an example, the cost of the labelling $f$ in the graph on the left side of Figure 1.12 can be improved from 16 to 11 by performing $R\langle\{p, q\}, l_3, f\rangle$.

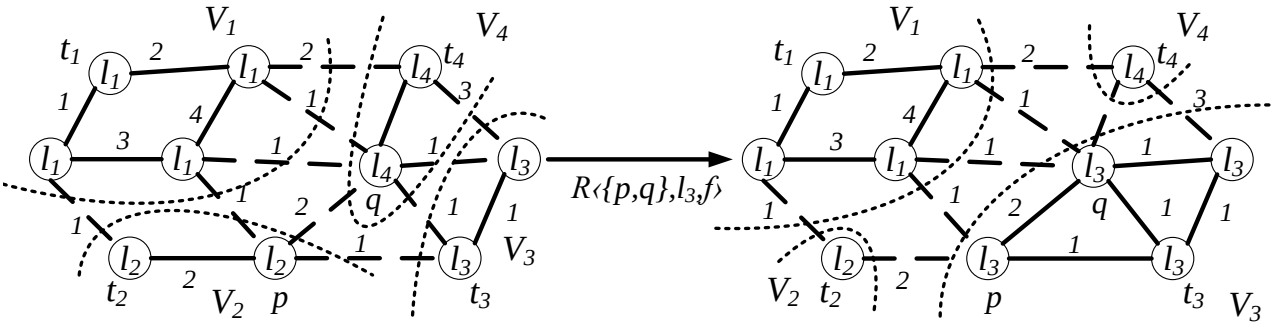


Figure 1.12: Example of a relabel operation $R\langle A, \alpha, f\rangle$, where $A = \{p, q\}$ and $\alpha = l_3$.

### 1.6.3 The Constrained Max $k$-cut Problem on Hypergraphs

Another well-known combinatorial optimization problem is the max $k$-cut problem. In the max $k$-cut problem we are given a weighted graph $G = (V, E)$ and an integer $k$, and the goal is to divide the set $V$ into $k$ non-empty partitions in such a way that the sum of the weights of the edges having their endpoints in different partitions is maximized. An instance of the max $k$-cut problem is shown in Figure 1.13 where $k = 4$. Partition $P = \{V_1, V_2, V_3, V_4\}$ is one possible solution to the problem; the dashed edges are the edges that contribute to the weight of the partition as shown in the figure.

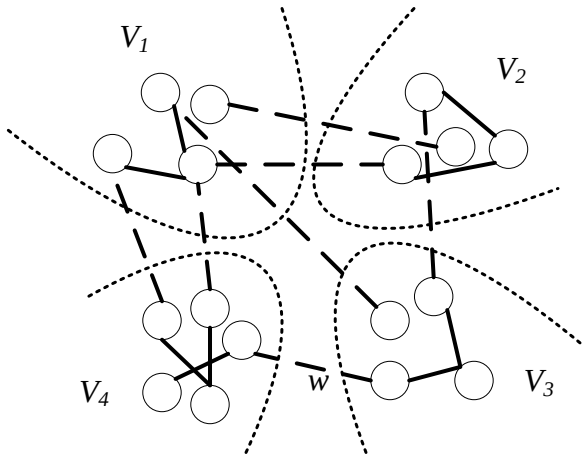


Figure 1.13: Instance of the max $k$-cut problem.

A hypergraph $H = (V, E)$ consist of a set $V$ of nodes and a set $E$ of hyperdges. A hyperedge $e$ consists of a set of nodes or *endpoints*. The size of a hyperedge $e$ is the number of its endpoints. Graphs are special cases of hypergraphs in which all the hyperedges have size 2.

In the max $k$-cut problem on hypergraphs the goal is to split the vertices into $k$ partitions in such a way that the sum of the weights of the hyperedges having at least two endpoints in different partitions is maximized. An instance of the max $k$-cut problem on hypergraphs where $k = 3$ is shown in Figure 1.14. Let $(u_1, u_2, \ldots, u_r)$ denote hyperedge $e$ where $u_1, u_2, \ldots, u_r$ are its endpoints. The hypergraph $H$ in Figure 1.14 consist of the following hyperedges: $e_1 = (v_1, v_2)$, $e_2 = (v_2, v_3, v_4)$, $e_3 = (v_1, v_5)$, $e_4 = (v_4, v_6, v_7, v_8)$ and $e_5 = (v_4, v_5, v_8)$. Let the weights of the hyperedges $e_1$, $e_2$, $e_3$, $e_4$, $e_5$ are 2, 3, 1, 4, 1 respectively. The cost of the partition $P = \{V_1, V_2, V_3\}$ shown in Figure 1.14 is 9, since each one of the hyperedges $e_2$, $e_3$, $e_4$ and $e_5$ have at least two endpoints located in different partitions.

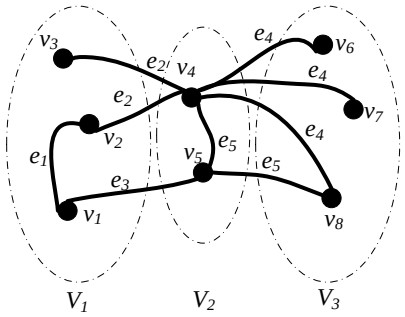


Figure 1.14: Instance of the max $k$-cut problem on hypergraphs.

The constrained max $k$-cut problem on hypergraph is a generalization of the max $k$-cut problem on hypergraphs in which there is an additional set $c$ of constraints that

each feasible solution must satisfy. We designed a local search approximation algorithm for this problem that uses two types of local operations: Move one node $u$ from one partition to another, and swap a node $u$ from some partition with a node $v$ in a different partition.

An example of these local operations is shown in Figure 1.15. In Figures 1.15 $a_1$, $a_2$ node $u$ in partition $V_i$ is moved to a different partition $V_l$, and in Figures 1.15 $b_1$, $b_2$ node $u$ in partition $V_i$ is swapped with node $v$ in partition $V_l$. Observe that by performing the move operation the costs of all the edges incident on $u$ with their other endpoints in $V_i$ are added to the cost of the solution (see Figure 1.15 $(a_2)$), while the costs of all the edges incident on $u$ with their other endpoints in $V_l$ no longer contribute to the cost of solution (see Figure 1.15 $(a_1)$). By performing the swap operation, the costs of all the edges incident on $u$ with their other endpoints in $V_i$ (see Figure 1.15 $(b_2)$) plus those of all the edges incident on $v$ with their other endpoints in $V_l$ (see Figure 1.15 $(b_2)$) are added to the cost of the solution, while the costs of all the edges incident on $u$ with their other endpoints in $V_l$ (see Figure 1.15 $(b_1)$) and the costs of all the edges incident on $v$ with their other endpoints in $V_i$ (see Figure 1.15 $(b_1)$) no longer contribute to the cost of solution.

Let $P = V_1, V_2, \ldots, V_k$ be any partition of the set of vertices. The neighborhood of $P$ includes all the partitions that can be obtained from $P$ by performing the two above local operations.

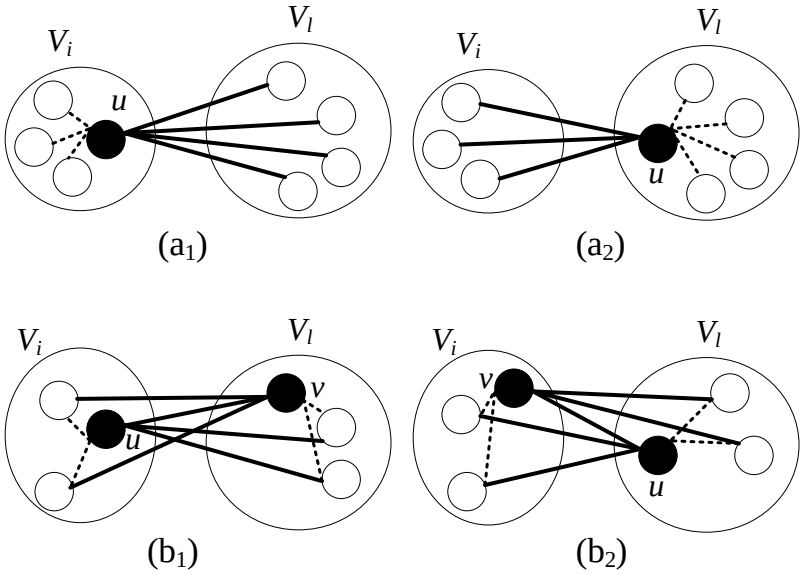


Figure 1.15: Example of local operations. The dashed edges in Figures $(a_1)$ and $(b_1)$ are the edges that are added to the cost of the solution after performing the corresponding local operation. The dashed edges in Figures $(a_2)$ and $(b_2)$ are the edges that no longer contribute to the cost of the solution after performing the local operation.

## 1.7 Organization of the Thesis

In Chapter 2 we include an article titled "Analysis of a local search algorithm for the $k$-facility location problem" published in the journal *RAIRO-Theoretical Informatics and Appplications* [21]. In this paper we present a local search algorithm for the metric $k$-facility location problem and give two different bounds for the locality gap of the algorithm, one matching the best known locality gap for the problem and a second one that is better for many cases.

In Chapter 3 we include an article titled "A local search algorithm for the multiway cut problem" that is currently under review in *Journal of Computer and System Sciences*. In this paper we present a local search algorithm for the mulitway cut problem and for two variations of the problem that arise from image processing applications. Moreover, we present a experimental comparison of the performance of our local search algorithm with that of algorithms with the currently best known approximation ratios for the multiway cut problem. Our experiments show that our algorithm has comparable performance to those algorithms, but is conceptually much simpler.

In Chapter 4 we include an article titled "A local search algorithm for the constrained max $k$-cut problem on hypergraphs" that is currently under review in *Journal of Applied Mathematics and Computing*. In this paper we present local search algorithms for the constrained max $k$-cut problem on hypergraphs. The constrained max $k$-cut problem is a generalization of the following problems: Max multiway cut, max Steiner $k$-cut, capacitated max $k$-cut, max $k$-cut with given sizes of parts, and directed max $k$-cut.

In Chapter 5 we summarize the main contributions of our research and discuss the approach we used to deal with the challenges of using local search in the design of approximation algorithms; we also discuss the strengths of local search algorithms.

# Bibliography


[1] E. Angel, and V. Zissimopoulos, On the landscape ruggedness of the quadratic assignment problem. *Theoretical Computer Science*, 263, (2001), 159-172.

[2] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, Complexity and approximation: Combinatorial optimization problems and their approximability properties. *Springer Science and Business Media*, (2012).

[3] P. Brucker, J. Hurink, and F. Werner, Improving local search heuristics for some scheduling problems II, *Discrete Applied Mathematics*, 72, (1997), 47-69.

[4] I. Boussäid, J. Lepagnot, and P. Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237, (2013), 82-117.

[5] B. Chandra and M. M. Halldörsson, Greedy local improvement and weighted set packing approximation, *Journal of Algorithms* 39, (2001), 223-240.

[6] B. Chandra, H. Karloff, and C. Tovey, New results on the old $k$-opt algorithm for the traveling salesman problem. *SIAM Journal on Computing*, 28, (1999), 1998-2029.

[7] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, M. Yannakakis, The complexity of multiterminal cuts. *SIAM Journal on Computing 23,* (1994), 864-894.

[8] R. C. Duh, and M. Fürer, Approximation of k-set cover by semi-local optimization. *In Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing,* (1997), 256-264.

[9] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey, An analysis of approximations for finding a maximum weight Hamiltonian circuit. *Operations Research*, 27, (1979), 799-809.

[10] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly,* 69(1), (1962), 9-15.

[11] M. R. Garey and D. S. Johnson, Computers and Intractability, *Freeman*, New York, (1979).

[12] R.L. Graham, Bounds for certain multiprocessor anomalies, *Bell System Technical Journal*, 45, (1966), 1563-1581.

[13] M. M. Halldörsson, and H. C. Lau, Low-Degree Graph Partitioning via Local Search with Applications to Constraint Satisfaction, Max-Cut and 3-Colouring. *Journal of Graph Algorithms and Applications*, 1, (1997), 1-13.

[14] P. Hansen and B. Jaumard, Algorithms for the maximum satisfiability problem. *Computing*, 44, (1990), 279-303.

[15] K. Iwama, S. Miyazaki, and N. Yamauchi, A 1.875 approximation algorithm for the stable marriage problem. *In Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, (2007), 288-297.

[16] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37, (1988), 79-100.

[17] S. Khuller, R. Bhatia, and R. Pless, On local search and placement of meters in networks. *SIAM Journal on Computing*, 32, (2003), 470-487.

[18] H. Lu and R. Ravi, The power of local optimization: approximation algorithms for maximum-leaf spanning tree, *Proceedings of the Thirtieth Annual Allerton Conference on Communication, Control, and Computing*, (1992), 533-542.

[19] J. B. Orlin, A. P. Punnen, A. S. Schulz. Approximate local search in combinatorial optimization. *SIAM Journal on Computing*, 33(5), (2004), 1201-1214.

[20] S. Sahni and T. Gonzales. P-complete approximation problems. *Journal of the ACM*, 23, (1976), 555-565.

[21] N. Samei and R. Solis-Oba, Analysis of a local search algorithm for the k-facility location problem. *RAIRO-Theoretical Informatics and Applications*, 49(4), (2015), 285-306.

# Chapter 2

# Analysis of a Local Search Algorithm for the $k$-Facility Location Problem

## 2.1 Introduction

In the $k$-facility location problem we are given set $F$ of facilities, a set $C$ of clients and an integer value $k > 0$. Each facility $j \in F$ has an opening cost $f_j$ and if facility $j$ is opened it can serve client $i \in C$ at cost $c_{ij}$. The goal is to select a subset $S$ of at most $k$ facilities that minimizes the cost of serving all the clients plus the cost of opening the facilities in $S$. Let $\sigma(j)$ represent the facility in $S$ that serves client $j$; then the goal is to minimize

$$cost(S) = \Sigma_{i \in S} f_i + \Sigma_{j \in C} c_{j\sigma(j)}.$$

Let $cost_s(S) = \Sigma_{j \in C} c_{j\sigma(j)}$ and $cost_f(S) = \Sigma_{i \in S} f_i$ denote the service cost and facility cost of solution $S$ respectively; then $cost(S) = cost_s(S) + cost_f(S)$.

If the service costs satisfy the triangle inequality, the problem is known as the metric $k$-facility location problem. If we eliminate the constraint on the number of facilities, the problem is called the facility location problem. Another special case of the $k$-facility location problem is when all the facility costs are zero, then the problem is known as the $k$-median problem. The $k$-facility location problem and its variants have applications in a large number of areas, such as banking [6], distributed systems [10], web services [12] and network design [2].

The facility location, $k$-median and $k$-facility location problems are known to be NP-hard. Therefore, extensive research has been done on designing approximation algorithms for these problems. For the metric $k$-facility location problem Jain and Vazirani [8] obtained a 6-approximation algorithm using a primal-dual technique; this approximation ratio was improved to 4 by Jain et al. [9] using a dual fitting technique, and later Zhang [14] used a local search approach to improve the approximation ratio to $2+\sqrt{3}+\epsilon$ for any constant $\epsilon > 0$. For the metric facility location problem Shmoys, Tardos and Aardal [13] obtained the first constant approximation algorithm by using a linear programming-based technique. Jain and Vazirani [8] obtained a better result using a primal-dual technique

yielding an algorithm with approximation ratio 3. The currently best known algorithm for the problem is by Li [11] with approximation ratio 1.488. For the metric $k$-median problem Charikar et al. [5] used a linear programming-based technique to design the first constant ratio approximation algorithm. Charikar and Guha [3] combined a primal-dual technique with a greedy approach and designed an improved algorithm with approximation ratio 4. Arya et al. [1] utilized a local search heuristic to design an algorithm with approximation ratio $3 + \epsilon$ for any constant $\epsilon > 0$.

### 2.1.1 Contributions

In this paper we focus on the metric $k$-facility location problem and show that a local search approach in which the only allowed operation is multi-swaps, where we can simultaneously swap $p \geq 1$ facilities in the solution with $p$ facilities not in the solution, has approximation ratio $max\{3,\, 5 - 2\frac{p-1}{q-1}\}$, where $q$ is a parameter whose value depends on the instance and it will be defined in Section 2.3.2. For those instances when $q$ is close to $p$ the approximation ratio is close to 3. We present an example showing the tightness of our bound. Using scaling [4] we get a second bound for the approximation ratio of our local search algorithm. This bound, $2 + \frac{1}{p} + \sqrt{3 + \frac{2}{p} + \frac{1}{p^2}}$, matches the bound of the local search algorithm of Zhang [14], which uses insertions and deletions in addition to swaps.

Our local search model is simpler than the one used by Zhang, as it uses only swaps; therefore, our algorithm always considers solutions of the same size, and as a result the search space that our algorithm explores is smaller than the one defined by Zhang's model. It is interesting that our algorithm with its more restricted set of operations achieves the same performance as that of Zhang's algorithm which makes use of a richer set of operations. As a result of using a simpler model our analysis is also simpler than that in [14]. Furthermore, with some minor changes our algorithm could find approximate solutions with the same above approximation ratio for all instances of the $k'$-facility location problem with $k' < k$. Our first bound is better than the second one when $q \leq (1 + \frac{\sqrt{3}}{3})p - \frac{\sqrt{3}}{3}$. In addition, for the special case when the ratio of the largest facility cost to the smallest facility cost is less than $p + 1$ our first bound reduces to $3 + \frac{2}{p}$ the same approximation ratio as that of the algorithm of Arya et al. [1] for the $k$-median problem.

### 2.1.2 Organization of the Paper

The rest of the paper is organized in the following way. In Section 2.2 we propose a local search algorithm for the $k$-facility location problem that uses multi-swap operations. In Section 2.3 we analyse the local optimal solutions produced by our algorithm and compute the first upper bound for its approximation ratio. In Section 2.4 we present an example showing the tightness of the bound. In Section 2.5 we present a different analysis of the algorithm and show that its approximation ratio matches that of Zhang's algorithm.

## 2.2 A Local Search Algorithm with Multiple Swaps

Let $S$ be a set of at most $k$ facilities. We present below a local search algorithm for the metric $k$-facility location problem based on the following multi-swap operation:

`swap <A,B> :=` $(S \setminus A) \cup B$

where $A \subseteq S$, $B \subseteq F \setminus S$, and $|A| = |B| \leq p$, for a constant $p \geq 1$.

**Algorithm 1** Local-Search ($F$, $C$, $k$)

1: **Input:** Set $F$ of facilities, set $C$ of clients and integer value $k$
2: **Output:** Local optimum solution
3: $S \leftarrow$ any set of $k$ facilities from $F$
4:
5: **for** $i \leftarrow 1$ **to** $k$ **do**
6: $S' \leftarrow$ any subset of $i$ facilities from $F$
7:
8: **while** $\exists$ a multi-swap operation $\langle A, B \rangle$ for $S'$ such that $cost((S' \setminus A) \cup B) < cost(S')$ **do**
9: $S' \leftarrow (S' \setminus A) \cup B$
10: **end while**
11: **if** $cost(S) > cost(S')$ **then**
12: $S \leftarrow S'$
13: **end if**
14: **end for**
15: return $S$

In steps 8-10 algorithm Local-Search repeatedly tries to improve on the current solution $S'$ by performing multi-swap operations. This process continues until no multi-swap operation can further improve the cost of the solution; therefore the algorithm finds a local optimal solution of size $i$ for each $1 \leq i \leq k$. At the end the algorithm returns the local optimal solution $S$ of minimum cost.

Let $S^*$ be an optimal solution, where $|S^*| = l \leq k$. Let $S^f$ be the final set of facilities selected by Local Search and let $S_i$ be the set selected by the algorithm for each $i = 1, ..., k$. Since $cost(S^f) \leq cost(S_l)$, if we could prove that $cost(S_l) \leq \alpha\, cost(S^*)$ for some value $\alpha$ then we would have proven that $cost(S^f) \leq \alpha\, cost(S^*)$ thus showing that the approximation ratio of algorithm Local-Search is $\alpha$. We, of course, do not know the value of $l$ and that is why the "for" loop in the algorithm tries all possible values for $l$. We show that for all integers $i = 1, ..., k$ there exists a value $\alpha > 0$ for which $cost(S_i) \leq \alpha\, cost(S_i^*)$, where $S_i^*$ is an optimal solution that uses $i$ facilities. This will prove that $cost(S_l) \leq \alpha\, cost(S_l^*) = \alpha\, cost(S^*)$, and so $cost(S^f) \leq \alpha\, cost(S^*)$. Therefore, without loss of generality, in the sequel we analyse only the case when the local optimal solution and global optimal solution have the same size.

The *locality gap* of a local search algorithm for a minimization problem is defined as

the maximum ratio of the value of any local optimum solution produced by the algorithm to the corresponding global optimum value. The locality gap of Local-Search is then equal to its approximation ratio.

To compute the locality gap of algorithm Local-Search we consider a set $Q$ of swap operations involving the facilities in the local optimum solution $S$ and facilities from a global optimum solution $S^*$. Since $S$ is a local optimum solution, then for each multi-swap operation $\langle A_i, B_i\rangle \in Q$, where $A_i \subseteq S$ and $B_i \subseteq S^*$,

$$cost((S \setminus A_i) \cup B_i) \geq cost(S). \tag{2.1}$$

Hence, for each $s \in S$ and $o \in S^*$,

$$cost((S \setminus s) \cup o) \geq cost(S), \text{ and so} \tag{2.2}$$

$$cost((S \setminus s) \cup o) - cost(S) \geq 0 \tag{2.3}$$

Note that algorithm Local-Search might not run in time that is polynomial in the size of the input as every iteration of the "while" loop might only provide a marginal improvement in the cost of the solution so it might require a very large number of iterations to find a local optimum solution. We can proceed as in [1] to ensure a polynomial running time: Replace the condition of the "while" loop as follows:

**while** $\exists$ a multi-swap operation $\langle A, B\rangle$ for $S'$ such that $cost((S' \setminus A) \cup B) \leq (1 - \frac{\epsilon}{|Q|})cost(S')$ **do**

where $\epsilon > 0$ is a constant. Every iteration of this loop decreases the value of the solution by at least a factor of $\frac{\epsilon}{|Q|}$, hence the total number of iterations is at most

$$\frac{log(cost(S)) - log(cost(S^*))}{log(\frac{|Q|}{|Q|-\epsilon})} \leq \frac{log(k) + log(f_{max}) + log(n) + log(c_{max})}{log(\frac{|Q|}{|Q|-\epsilon})}$$

where $f_{max} = max\{f_i \mid i \in F\}$, $n = |C|$ and $c_{max} = max\{c_{ji} \mid i \in F, j \in C\}$. The above inequality holds because $cost(S) \leq k\, f_{max} + n\, c_{max}$ and without loss of generality we can assume $cost(S^*) \geq 1$.

The set $Q$ that we consider contains no more than $k^2 + k$ multi-swap operations as explained in Sections 2.3.2 and 2.5.2, so the total number of iterations performed by Local-Search is at most
$[log(k) + log(f_{max}) + log(n) + log(c_{max})]\, /log\left(1 + \frac{\epsilon}{k^2+k-\epsilon}\right) <$
$[log(k) + log(f_{max}) + log(n) + log(c_{max})]\, (2k^2 + k)/\epsilon.$

Each iteration of the "while" loop needs to consider at most $(k|F|)^p$ different sets $A$ and $B$, which is polynomial for $p$ constant. Therefore, the time complexity of the algorithm is polynomial in the size of the input. In the following sections we will show that

$$0 \leq \sum_{\langle A_i, B_i\rangle \in Q} [cost((S \setminus A_i) \cup B_i) - cost(S)] \leq \alpha\, cost(S^*) - cost(S) \tag{2.4}$$

for some constant $\alpha$. Therefore, the locality gap of algorithm Local-Search is $\alpha$. Hence, note that with the new termination condition of the "while" loop we could not use

inequality (2.4) to bound the locality gap, as the modified algorithm would not produce a local optimum solution, but only a solution $S$ for which for any $A \subseteq S$ and $B \subseteq F$, $cost((S \setminus A) \cup B) > \left(1 - \frac{\epsilon}{|Q|}\right) cost(S)$.

Observe that if we can prove (2.4) for any local optimum solution $S'$ then $cost(S') \leq \alpha\, cost(S^*)$. However, for the solution $S$ obtained by the modified algorithm we know that $cost((S \setminus A_i) \cup B_i) > \left(1 - \frac{\epsilon}{|Q|}\right) cost(S)$ for each $\langle A_i, B_i\rangle \in Q$; therefore, we have to modify (2.4) as follows:

$$\alpha\, cost(S^*) - cost(S) \geq \sum_{\langle A_i,B_i\rangle \in Q} [cost((S \setminus A_i) \cup B_i) - cost(S)] > -\frac{\epsilon}{|Q|} \sum_{\langle A_i,B_i\rangle \in Q} cost(S) = -\epsilon\, cost(S).$$

Hence $cost(S) \leq \frac{\alpha}{1-\epsilon} cost(S^*)$.

In the rest of the paper we will prove inequality (2.4) for any local optimum solution and some value $\alpha$. Then, by the above argument we will have proven that our algorithm with the modified "while" condition has approximation ratio $\frac{\alpha}{1-\epsilon}$.

## 2.3 First Bound for the Locality Gap

Since in a local optimum solution $S$ no multi-swap operation can improve its cost, then for any multi-swap operation $\langle A, B\rangle$ the following inequality is satisfied:

$$cost((S \setminus A) \cup B) \geq cost(S). \tag{2.5}$$

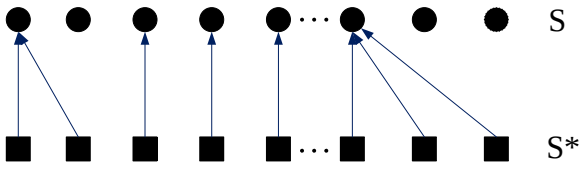


Figure 2.1: Mapping $\pi$ maps each $o \in S^*$ to its closest facility $\pi(o) \in S$.

We define a mapping $\pi$ similar as the mapping $\eta$ in [7] as follows: $\pi : S^* \to S$ maps each facility $o$ in the optimum solution to its closest facility $\pi(o) \in S$ breaking ties arbitrarily; hence, $c_{o\pi(o)} \leq c_{os}$ for all $s \in S$. The map $\pi$ can be represented as a bipartite graph as shown in Figure 2.1. For each facility $s \in S$, let the (in-)degree of $s$ in this bipartite graph be $deg(s) = |\pi^{-1}(s)|$. If $deg(s) \geq 2$ then we call $s$ a *bad facility* otherwise we call it a *good facility*.

### 2.3.1 Pairing

To bound the cost of the local optimum solution $S$ produced by our algorithm we will use several sets of multi-swap operations involving facilities from $S$ and facilities from $S^*$. These sets of multi-swap operations are chosen so that combining the local optimality condition (2.5) for all of them allows us to bound the cost of $S$ in terms of the cost of $S^*$. To this end we first use algorithm Partition below to divide $S$ and $S^*$ into subsets of facilities that will participate in the swaps. More specifically, $S$ and $S^*$ are partitioned into sets $A_1, A_2, ..., A_r$ and $B_1, B_2, ..., B_r$ respectively, where $|A_i| = |B_i|$ for all $1 \leq i \leq r$ and $r-1$ is equal to the number of bad facilities.

**Algorithm 2** Partition $(S, S^*)$

1: **Input:** Local optimum solution $S$ and global optimum solution $S^*$
2: **Output:** Partition $(A_1, A_2, ..., A_r)$ of $S$ and $(B_1, B_2, ..., B_r)$ of $S^*$
3: **for** $i \leftarrow 1$ **to** $r-1$ **do**
4: $A_i \leftarrow \{b\} \cup \{\text{any } deg(b)-1 \text{ facilities of } S \text{ with degree } 0\}$, where $b \in S$ is any bad facility
5: $B_i \leftarrow \pi^{-1}(b)$
6: $S \leftarrow S \setminus A_i$
7: $S^* \leftarrow S^* \setminus B_i$
8: **end for**
9: $A_r \leftarrow S$
10: $B_r \leftarrow S^*$
11: **return** $(A_1, A_2, ..., A_r)$, $(B_1, B_2, ..., B_r)$

Note the following facts:

I. In Step 4 there are enough facilities $s$ with degree 0 since $|S| = |S^*|$.

II. For any sets $A_i$ and $B_i$, where $1 \leq i < r$ if $o \in S^* \setminus B_i$, then $\pi(o) \notin A_i$.

III. For each facility $s \in A_r$, $\pi^{-1}(s) = o \in B_r$. To see this note that for each facility $o \in B_r$ it must be that $\pi(o) \in A_r$ because if $\pi(o) \in A_i$ for $i \neq r$, then $o$ would belong to $B_i$. Since $|A_r| = |B_r|$ and $A_r$ only includes good facilities, then for all $s \in A_r$, $deg(s) = 1$ and so $\pi^{-1}(s) \in B_r$.

We pair the facilities in $S$ with those in $S^*$ as follows:

- For $1 \leq i \leq r-1$ each set $A_i$ is paired with set $B_i$.
- Each facility $s \in A_r$ is paired with $o \in B_r$, where $o = \pi^{-1}(s)$.

Note that if we consider multi-swaps $\langle A_i, B_i \rangle$ for $1 \leq i \leq r-1$ and single swaps $\langle s, \pi^{-1}(s) \rangle$ for all $s \in A_r$, then each facility in $S$ and $S^*$ would participate in one swap operation and adding all inequalities (2.5) for these swaps would allow us to bound the cost of $S$ in terms of the cost of $S^*$. However, since we are only allowed to swap at most $p$ facilities simultaneously, then the above multi-swap operations would not be allowed

for those sets $A_i$ and $B_i$ whose size exceeds $p$. Therefore, we need to consider a different approach for these pairs. Algorithm Partition-2 further splits those sets $A_i$ and $B_i$, where $|A_i| = |B_i| > p$, and constructs *core subsets* $\hat{A}_i \subset A_i$ and $\hat{B}_i \subset B_i$, where $|\hat{A}_i| = |\hat{B}_i| = p$; $\hat{A}_i$ includes the bad facility in $A_i$ and $\hat{B}_i$ is built by finding a closest facility to each one of the facilities in $\hat{A}_i$.

**Algorithm 3** Partition-2 $(A_1, A_2, ..., A_r, B_1, B_2, ..., B_r)$

**Input:** Partition $(A_1, A_2, ..., A_r)$ of $S$ and $(B_1, B_2, ..., B_r)$ of $S^*$
**Output:** Core subsets $\hat{A}_i$ and $\hat{B}_i$ for all those sets $A_i$ and $B_i$ for which $|A_i| = |B_i| > p$
**for** $i \leftarrow 1$ **to** $r - 1$ **do**

  **if** $|A_i| > p$ **then**
    $\hat{A}_i \leftarrow \{b\} \cup \{\text{any } p-1 \text{ facilities of } A_i - b\}$, where $b \in A_i$ is the bad facility in $A_i$
    $\hat{B}_i \leftarrow \{\}$
    $A'_i \leftarrow \hat{A}_i$

    **for** $i \leftarrow 1$ **to** $p$ **do**
      $a_i \leftarrow$ any facility from $A'_i$
      $b_i \leftarrow$ nearest facility to $a_i$, where $b_i \in B_i$
      $\hat{B}_i \leftarrow \{b_i\} \cup \hat{B}_i$
      $A'_i \leftarrow A'_i \setminus \{a_i\}$
    **end for**
  **end if**
**end for**
**return** $(\hat{A}_1, \hat{A}_2, ..., \hat{A}_{r-1})$, $(\hat{B}_1, \hat{B}_2, ..., \hat{B}_{r-1})$

For sets $A_i$ and $B_i$, where $1 \leq i < r$ and $|A_i| = |B_i| > p$, we pair their facilities as follows:

- $\hat{A}_i$ is paired with $\hat{B}_i$.
- Each facility in $A_i \setminus \hat{A}_i$ is paired with a facility in $B_i \setminus \hat{B}_i$ so that each facility is paired once.

In the following sections, we will bound the cost of the local optimum solution $S$ produced by our algorithm in terms of the cost of a global optimum solution $S^*$ by considering swap operations involving the above pairs of facilities.

**The idea behind the pairings.** As mentioned above $\pi$ maps each facility in $S^*$ to its closest facility in $S$. To get some intuition as to why this is done, consider that facility $s \in S$ is close to exactly one facility $o \in S^*$ and far away from rest of facilities in $S^*$. Then if we swap $s$ and $o$, we close facility $s$ and open facility $o$ reassigning the clients of $s$ to $o$; this does not change the cost the solution too much, which means that the contribution of facility $s$ to the cost of the solution is similar to the contribution of facility $o$ to the cost

of the optimum solution. However, if $s$ is close to several facilities in $S^*$ then performing a swap between $s$ and the closest facility $o \in S^*$ might suggest a large difference between the cost of $S$ and the cost of $(S \setminus \{s\}) \cup \{o\}$ because re-assigning a client $j$ of $s$ to $o$ might have a much larger cost than assigning $j$ to its closest facility in $S^*$. That is the reason why we call the facilities in $S$ with degree larger than one bad and others "good" and why a "bad" facility $b$ is not swapped in our analysis with a single facility from $S^*$, but instead a set of facilities containing $b$ is swapped with a set of nearby facilities from $S^*$.

### 2.3.2 Analysing the Swaps

Let $s_j = c_{j\sigma(j)}$ and $o_j = c_{j\sigma^*(j)}$ be the service costs of client $j$ in solutions $S$ and $S^*$ respectively, where $\sigma(j)$ is the facility closest to $j$ in $S$ and $\sigma^*(j)$ is the facility closest to $j$ in $S^*$. Let $N_S(s) = \{j \mid \sigma(j) = s\}$ be the set of clients that are served by facility $s$ in the local optimal solution and $N_S^*(o) = \{j \mid \sigma^*(j) = o\}$ is the set of clients that are served by $o$ in the global optimal solution. We extend these definitions to sets $A_i \subseteq S$ and $B_i \subseteq S^*$, so $N_S(A_i)$ is the set of clients that are served by facilities in $A_i$ in $S$ and $N_S^*(B_i)$ are those clients that are served by $B_i$ in $S^*$.

All the lemmas of this section are about bounding the cost increase caused by a swap operation $\langle A, B\rangle$, where $A \subseteq S$ and $B \subseteq F \setminus S$, and they are all based on the fact that $cost((S \setminus A) \cup B) - cost(S) \geq 0$. In these lemmas we introduce a new assignment of client to facilities after the swap $\langle A, B\rangle$ is performed and bound the service cost of this assignment. More specifically, some of the clients $j$ are re-assigned to $\pi(\sigma^*(j))$. The following lemma bounds the cost of serving client $j$ by $\pi(\sigma^*(j))$.

**Lemma 1** *(Cost Bounding)* $c_{j\pi(\sigma^*(j))} \leq 2o_j + s_j$.

**Proof**

$$\begin{aligned} c_{j\pi(\sigma^*(j))} &\leq c_{j\sigma^*(j)} + c_{\sigma^*(j)\pi(\sigma^*(j))} && (2.6)\\ &\leq c_{j\sigma^*(j)} + c_{\sigma^*(j)\sigma(j)} && (2.7)\\ &\leq c_{j\sigma^*(j)} + c_{j\sigma^*(j)} + c_{j\sigma(j)} = 2o_j + s_j. && (2.8)\end{aligned}$$

Inequalities (2.6) and (2.8) follow from the triangle inequality and (2.7) is true since $\pi(\sigma^*(j))$ is the nearest facility in $S$ to $\sigma^*(j)$ (see Figure 2.2).

**Multi-Swaps for Sets $A_i$ and $B_i$ where $|A_i| = |B_i| \leq p$**

**Lemma 2** *For each swap $\langle A_i, B_i\rangle$ where $|A_i| = |B_i| \leq p$ and $1 \leq i < r$,*

$$\sum_{o \in B_i} f_o - \sum_{s \in A_i} f_s + \sum_{j \in N_{S^*}(B_i)} (o_j - s_j) + \sum_{j \in N_S(A_i)} 2o_j \geq 0. \qquad (2.9)$$

**Proof** By performing swap $\langle A_i, B_i\rangle$, facilities in $A_i$ are closed and those in $B_i$ are opened. Therefore, the clients $j \in N_S(A_i)$ need to be re-assigned to the facilities in $(S \setminus A_i) \cup B_i$. To bound the cost of the new solution let us consider the following assignment of clients to facilities:

1. Assign all the clients in $N_S^*(B_i)$ to facilities in $B_i$ in the same way in which they are assigned in $S^*$.

2. Assign each client $j$ in $N_S(A_i) \setminus N_S^*(B_i)$ to $\hat{s} = \pi(o)$, where $o = \sigma^*(j)$ is the facility closest to $j$ in $S^*$. Note that $\pi(o)$ is the closest facility to $o$ in $S$ and $\pi(o) \notin A_i$ by Fact II in Section 2.3.1 (see Figure 2.2).

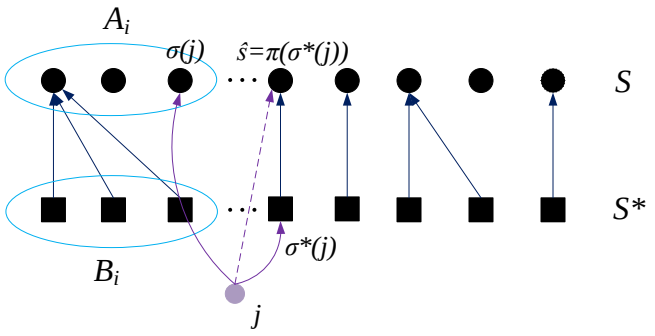


Figure 2.2: Client $j \in N_S(A_i) \setminus N_S^*(B_i)$ is assigned to $\hat{s} = \pi(\sigma^*(j))$. Note that $\sigma^*(j) \notin B_i$ and $\pi(\sigma^*(j)) \notin A_i$.

3. The assignment of all other clients to facilities remains unchanged.

The difference in cost between solution $S$ and $(S \setminus A_i) \cup B_i$ caused by re-assigning client $j \in N_S^*(B_i)$ to $\sigma^*(j)$ is $o_j - s_j$. Adding these cost changes over all clients in $N_S^*(B_i)$ we obtain the third term in (2.9). For clients $j \in N_S(A_i) \setminus N_S^*(B_i)$ using Lemma 1 the cost change is bounded by $2o_j$. Adding these changes over all clients in $N_S(A_i) \setminus N_S^*(B_i)$ gives $\sum_{j \in N_S(A_i) \setminus N_S^*(B_i)} 2o_j$. The fourth term in (2.9) is obtained by considering the fact that $\sum_{j \in N_S(A_i) \setminus N_S^*(B_i)} 2o_j \leq \sum_{j \in N_S(A_i)} 2o_j$. Finally, the first two terms in (2.9) are the result of adding the costs of the opened facilities in $B_i$ and subtracting the costs of the closed facilities in $A_i$.

**Swaps for Sets $A_i$ and $B_i$ where $|A_i| = |B_i| > p$**

If $|A_i| = |B_i| = q_i > p$, we perform three different sets of swaps involving these facilities. First, we perform the swap $\langle \hat{A}_i, \hat{B}_i \rangle$.

**Lemma 3** *For each swap $\langle \hat{A}_i, \hat{B}_i \rangle$,*

$$\sum_{o \in \hat{B}_i} f_o - \sum_{s \in \hat{A}_i} f_s + \sum_{j \in N_{S^*}(\hat{B}_i)} (o_j - s_j) + \sum_{\substack{j \in N_S(\hat{A}_i) \setminus N_{S^*}(\hat{B}_i) \\ \pi(\sigma^*(j)) \in \hat{A}_i}} (o_j + s_j) + \sum_{\substack{j \in N_S(\hat{A}_i) \setminus N_{S^*}(\hat{B}_i) \\ \pi(\sigma^*(j)) \notin \hat{A}_i}} 2o_j \geq 0. \qquad (2.10)$$

**Proof** Since $\hat{A}_i$ and $\hat{B}_i$ are subsets of $A_i$ and $B_i$ respectively, then Fact II in Section 2.3.1 might not hold for them. In other words for some clients $j \in N_S(\hat{A}_i) \setminus N_S^*(\hat{B}_i)$ it might be that $\pi(\sigma^*(j)) \in \hat{A}_i$. Therefore, if we want to proceed similarly as in the proof of Lemma 2 we need to define a new re-assignment for clients $j \in N_S(\hat{A}_i) \setminus N_S^*(\hat{B}_i)$ for which $\pi(\sigma^*(j)) \in \hat{A}_i$. Consider the following assignment of clients to facilities in $(S \setminus \hat{A}_i) \cup \hat{B}_i$:

1. Assign client $j \in N_S^*(\hat{B}_i)$ to $\sigma^*(j) \in \hat{B}_i$; this changes the cost of $S$ by $o_j - s_j$. Adding these cost changes over all clients in $N_S^*(\hat{B}_i)$ gives the third term in (2.10).

2. Assign each client $j$ in $N_S(\hat{A}_i) \setminus N_S^*(\hat{B}_i)$ such that $\pi(\sigma^*(j)) \notin \hat{A}_i$ to $\pi(\sigma^*(j))$. Using Lemma 1 the cost increase per client $j$ for this reassignment is $2o_j$. Adding these cost increases over all these clients gives us the fifth term in (2.10).

3. Consider a client $j \in N_S(\hat{A}_i) \setminus N_S^*(\hat{B}_i)$ for which $\pi(\sigma^*(j)) \in \hat{A}_i$. Let $s \in \hat{A}_i$ be the facility serving $j$ in $S$ and $o \in \hat{B}_i$ be the closest facility to $s$; client $j$ is assigned to $o$. Let $o'$ be the facility serving $j$ in $S^*$. The change in cost caused by reassigning client $j$ to $o$ is $c_{jo} - s_j$. Note that

$$\begin{aligned} c_{jo} - s_j &\leq c_{js} + c_{so} - s_j && (2.11)\\ &\leq c_{js} + c_{so'} - s_j && (2.12)\\ &\leq c_{js} + c_{js} + c_{jo'} - s_j = o_j + s_j. && (2.13) \end{aligned}$$

   Inequalities (2.11) and (2.13) hold because of the triangle inequality and (2.12) is true because $o$ is closer to $s$ than $o'$. Adding these cost increases over all client in $N_S(\hat{A}_i) \setminus N_S^*(\hat{B}_i)$ for which $\pi(\sigma^*(j)) \in (\hat{A}_i)$ gives us the fourth term in (2.10).

4. The assignment of the rest of the clients to facilities remains unchanged.

Finally, the first two terms in (2.10) are the result of adding the costs of all the opened facilities in $\hat{B}_i$ and subtracting the cost of the closed ones in $\hat{A}_i$.

Note that the fourth term in inequality (2.10) includes the service cost $s_j$ for some clients as a positive term. Since our goal is to find an upper bound for the cost of the local optimum solution $S$, the appearance of these positive service costs $s_j$ on the left side of (2.10) is problematic. To get rid of these terms we perform a second set of swaps for pairs $\langle s, o\rangle$, where $s \in A_i \setminus \hat{A}_i$ and $o \in B_i \setminus \hat{B}_i$.

**Corollary 2.3.1** *For each swap $\langle s, o\rangle$ where $s \in A_i \setminus \hat{A}_i$ and $o \in B_i \setminus \hat{B}_i$,*

$$f_o - f_s + \sum_{\substack{j \in N_{S^*}(o) \cap N_S(\hat{A}_i),\\ \pi(\sigma^*(j)) \in \hat{A}_i}} (o_j - s_j) + \sum_{j \in N_S(s)} 2o_j \geq 0. \tag{2.14}$$

$$f_o - f_s + \sum_{j \in N_{S^*}(o)} (o_j - s_j) + \sum_{j \in N_S(s)} (o_j + o_j + s_j - s_j) \geq 0. \tag{2.15}$$

**Proof** In Lemma 2 replace $A_i$ with $s$ and $B_i$ with $o$ and change the first and second re-assignment of clients to facilities as follows:

1. Assign all clients $j \in N_{S^*}(o) \cap N_S(\hat{A}_i)$ such that $\pi(\sigma^*(j)) \in \hat{A}_i$ to $o$.
2. Assign all clients $j \in N_S(s)$ to $\pi(\sigma^*(j))$. Note that this is a valid assignment because $s$ is a good facility with degree 0; therefore, $\pi(o) \neq s$ for every facility $o \in S^*$.

**Lemma 4** *For each $i = 1, ..., r-1$ such that $|A_i| = |B_i| > p$,*

$$\sum_{o \in B_i \setminus \hat{B}_i} f_o - \sum_{s \in A_i \setminus \hat{A}_i} f_s + \sum_{\substack{j \in N_S(\hat{A}_i) \setminus N_S^*(\hat{B}_i), \\ \pi(\sigma^*(j)) \in \hat{A}_i}} (o_j - s_j) + \sum_{j \in N_S(A_i) \setminus N_S(\hat{A}_i)} 2o_j \geq 0. \tag{2.16}$$

**Proof** By Fact II in Section 2.3.1 , for any sets $A_i$ and $B_i$, where $1 \leq i < r$, if $o \in S^* \setminus B_i$ then $\pi(o) \notin A_i$; thus, if $\pi(o) \in A_i$ then $o \in B_i$. Consequently, for a client $j \in N_S(\hat{A}_i)$ if $\pi(\sigma^*(j)) \in \hat{A}_i$ then $j$ belongs to $N_S^*(B_i)$; therefore, $\{j \mid j \in N_S(\hat{A}_i), \pi(\sigma^*(j)) \in \hat{A}_i\} \subseteq N_S^*(B_i)$ and so $\{j \mid j \in N_S^*(B_i) \cap N_S(\hat{A}_i), \pi(\sigma^*(j)) \in \hat{A}_i\} = \{j \mid j \in N_S(\hat{A}_i), \pi(\sigma^*(j)) \in \hat{A}_i\}$. Hence,

$$\begin{aligned} \{j \mid j \in [N_S^*(B_i) \setminus N_S^*(\hat{B}_i)] \cap N_S(\hat{A}_i), \pi(\sigma^*(j)) \in \hat{A}_i\} = \\ \{j \mid j \in [N_S^*(B_i) \cap N_S(\hat{A}_i)] \setminus N_S^*(\hat{B}_i), \pi(\sigma^*(j)) \in \hat{A}_i\} = \\ \{j \mid j \in N_S(\hat{A}_i) \setminus N_S^*(\hat{B}_i), \pi(\sigma^*(j)) \in \hat{A}_i\}. \end{aligned} \tag{2.17}$$

Adding inequality (2.15) over all facilities $o \in B_i \setminus \hat{B}_i$ and $s \in A_i \setminus \hat{A}_i$ and using (2.17) we get (2.16).

As mentioned before since we do not want the positive service cost $s_j$ in the left side of (2.10), we add (2.10) and (2.16) to discard the undesired terms

$$\sum_{o \in B_i} f_o - \sum_{s \in A_i} f_s + \sum_{j \in N_{S^*}(\hat{B}_i)} (o_j - s_j) + \sum_{j \in N_S(A_i)} 2o_j \geq 0. \tag{2.18}$$

To get the fourth term in (2.18) note that adding the third term in (2.16) and the fourth term in (2.10) we get

$$\sum_{\substack{j \in N_S(\hat{A}_i) \setminus N_S^*(\hat{B}_i), \\ \pi(\sigma^*(j)) \in \hat{A}_i}} (o_j - s_j) + \sum_{\substack{j \in N_S(\hat{A}_i) \setminus N_S^*(\hat{B}_i), \\ \pi(\sigma^*(j)) \in \hat{A}_i}} (o_j + s_j) = \sum_{\substack{j \in N_S(\hat{A}_i) \setminus N_S^*(\hat{B}_i), \\ \pi(\sigma^*(j)) \in \hat{A}_i}} 2o_j.$$

Adding the right hand side of the above equality and the fifth term in (2.10) yields

$$\sum_{\substack{j \in N_S(\hat{A}_i) \setminus N_S^*(\hat{B}_i), \\ \pi(\sigma^*(j)) \in \hat{A}_i}} 2o_j + \sum_{\substack{j \in N_S(\hat{A}_i) \setminus N_S^*(\hat{B}_i), \\ \pi(\sigma^*(j)) \notin \hat{A}_i}} 2o_j = \sum_{j \in N_S(\hat{A}_i) \setminus N_S^*(\hat{B}_i)} 2o_j.$$

Finally, adding the right hand side of the above equality and the fourth term in (2.16) we get

$$\sum_{j\in N_S(\hat{A}_i)\setminus N_S^*(\hat{B}_i)} 2o_j + \sum_{j\in N_S(A_i)\setminus N_S(\hat{A}_i)} 2o_j \le \sum_{j\in N_S(A_i)} 2o_j.$$

As our goal is to find an upper bound for $cost(S)$, we note that the left hand side of inequality (2.18) is missing the service cost of the clients that are served by facilities in $B_i \setminus \hat{B}_i$. To include this missing cost we perform a third set of swaps, where each good facility in $A_i$ is swapped with every facility in $B_i \setminus \hat{B}_i$.

**Corollary 2.3.2** *For each swap $\langle s, o\rangle$ where $s$ is a good facility in $A_i$ and $o \in B_i$,*

$$f_o - f_s + \sum_{j\in N_{S^*}(o)} (o_j - s_j) + \sum_{j\in N_S(s)} 2o_j \ge 0. \tag{2.19}$$

**Proof** In Lemma 2 replace $A_i$ with $s$ and $B_i$ with $o$ and note that assigning all clients $j \in N_S(s) \setminus N_{S^*}(o)$ to $\pi(\sigma^*(j))$ is a valid assignment because $s$ is a good facility and so $s \ne \pi(o)$ for all $o \in S^*$.

Let $b_i$ be the bad facility in $A_i$ and let $q_i = |A_i| = |B_i|$. Adding the inequality (2.19) for all pairs $\langle s, o\rangle \in (A_i - b_i) \times (B_i \setminus \hat{B}_i)$, we get

$$(q_i - 1)\sum_{o\in B_i\setminus\hat{B}_i} f_o - (q_i - p)\sum_{s\in A_i - b_i} f_s + (q_i - 1)\sum_{j\in N_S^*(B_i)\setminus N_S^*(\hat{B}_i)} (o_j - s_j) + (q_i - p)\sum_{j\in N_S(A_i - b_i)} 2o_j \ge 0. \tag{2.20}$$

Observe that each facility $o \in B_i$ is swapped $q_i - 1$ times, therefore facility cost $f_o$ and service cost change $o_j - s_j$ for clients $j \in N_{S^*}(B_i) \setminus N_{S^*}(\hat{B}_i)$ are added $q_i - 1$ times. In addition, each good facility $s \in A_i$ is swapped $q_i - p$ times, therefore facility cost $-f_s$ and service cost $2o_j$ (the fourth term in (2.19)) for clients $j \in N_S(s)$ are added $q_i - p$ times.

Multiplying (2.20) by $\frac{1}{q_i - 1}$ and adding to (2.18) we get

$$\begin{aligned}\sum_{o\in B_i} f_o + \sum_{o\in B_i\setminus\hat{B}_i} f_o - \sum_{s\in A_i} f_s - \frac{q_i - p}{q_i - 1}\sum_{s\in A_i - b_i} f_s \\ + \sum_{j\in N_{S^*}(B_i)} (o_j - s_j) + \sum_{j\in N_S(A_i)} 2o_j + \frac{q_i - p}{q_i - 1}\sum_{j\in N_S(A_i - b_i)} 2o_j \ge 0.\end{aligned} \tag{2.21}$$

Using $\sum_{o\in B_i\setminus B_i^*} f_o \le \sum_{o\in B_i} f_o$, $\frac{q_i - p}{q_i - 1}\sum_{s\in A_i - b_i} f_s > 0$ and $\sum_{j\in N_S(b_i)} 2o_j > 0$ in (2.21) we get

$$2\sum_{o\in B_i} f_o - \sum_{s\in A_i} f_s + \sum_{j\in N_{S^*}(B_i)} (o_j - s_j) + \left(2 - \frac{p-1}{q_i - 1}\right)\sum_{j\in N_S(A_i)} 2o_j \ge 0. \tag{2.22}$$

**Single Swaps for Facilities in Sets $A_r$ and $B_r$**

**Corollary 2.3.3** *For each swap $\langle s, o\rangle$, where $s \in A_r$ has been paired with $o \in B_r$,*

$$f_o - f_s + \sum_{j \in N_{S^*}(o)} (o_j - s_j) + \sum_{j \in N_S(s)} 2o_j \geq 0. \tag{2.23}$$

**Proof** In Lemma 2 replace $A_i$ with $s$ and $B_i$ with $o$ and note that assigning all clients $j \in N_S(s) \setminus N_{S^*}(o)$ to $\pi(\sigma^*(j))$ is a valid assignment since if $j \notin N_S(o)$ then $\pi(\sigma^*(j)) \neq s$ as $s = \pi(o)$ for each pair $(s, o)$ in sets $A_r$ and $B_r$.

Adding the inequalities (2.23) for all pairs $(s, o)$ in sets $A_r$ and $B_r$ we get

$$\sum_{o \in B_r} f_o - \sum_{s \in A_r} f_s + \sum_{j \in N_{S^*}(B_r)} (o_j - s_j) + \sum_{j \in N_S(A_r)} 2o_j \geq 0. \tag{2.24}$$

**Putting It All Together**

Let $G \subseteq S$ be the set of facilities in all sets $A_i$, $1 \leq i < r$ for which $|A_i| > p$. These facilities are swapped with some facilities in $S^*$ as explained in Section 2.3.2; let $G^* \subseteq S^*$ be this set of facilities. Let $I = \{i \mid 1 \leq i < r, |A_i| > p\}$ and $I^c = \{i \mid 1 \leq i < r, |A_i| \leq p\}$. Adding inequalities (2.9) for all sets $A_i$, $i \in I^c$, inequalities (2.22) for all sets $A_i$, $i \in I$, and inequality (2.24) we get

$$\sum_{o \in S^*} f_o + \sum_{o \in G^*} f_o - \sum_{s \in S} f_s + \sum_{j \in C} (o_j - s_j) + \sum_{j \in C \setminus N_S(G)} 2o_j + \sum_{i \in I} \left[ \left( 2 - \frac{p-1}{q_i - 1} \right) \sum_{j \in N_S(A_i)} 2o_j \right] \geq 0. \tag{2.25}$$

**Lemma 5**

$$\sum_{i \in I} \left[ \left( 2 - \frac{p-1}{q_i - 1} \right) \sum_{j \in N_S(A_i)} 2o_j \right] \leq \left( 2 - \frac{p-1}{q-1} \right) \sum_{j \in N_S(G)} 2o_j$$

*where $q = max\ \{q_i |\ i \in I\}$.*

**Proof** The lemma follows since $2o_j$ is positive.

Using Lemma 5 inequality (2.25) can be rewritten as follows:

$$\sum_{o \in S^*} f_o + \sum_{o \in G^*} f_o - \sum_{s \in S} f_s + \sum_{j \in C} (o_j - s_j) + \sum_{j \in C} 2o_j + \left( 1 - \frac{p-1}{q-1} \right) \sum_{j \in N_S(G)} 2o_j \geq 0. \tag{2.26}$$

The fifth term in (2.26) is obtained by adding $\sum_{j \in N_S(G)} 2o_j$ to the fifth term of (2.25) and the sixth term in (2.26) is obtained by subtracting $\sum_{j \in N_S(G)} 2o_j$ from the last term of (2.25).

Since $\sum_{j\in N_S(G)} 2o_j \leq \sum_{j\in C} 2o_j$, $G \subseteq S$, and all the facility and service costs are positive then

$$\begin{aligned} 0 &\leq 2\sum_{o\in S^*} f_o - \sum_{s\in S} f_s + \sum_{j\in C}(o_j - s_j) + \left(2 - \frac{p-1}{q-1}\right)\sum_{j\in C} 2o_j \\ &= 2cost_f(S^*) + \left[5 - 2\left(\frac{p-1}{q-1}\right)\right] cost_s(S^*) - cost_f(S) - cost_s(S). \end{aligned}$$

Therefore, since $p < q$ and so $\left[5 - 2\left(\frac{p-1}{q-1}\right)\right] > 2$, then

$$\left[5 - 2\left(\frac{p-1}{q-1}\right)\right](cost_f(S^*) + cost_s(S^*)) \geq cost_f(S) + cost_s(S).$$

Notice that if no set $A_i$ has size larger than $p$ then the swaps considered in Section 2.3.2 are not needed and in that case it can be shown that

$$3(cost_f(S^*) + cost_s(S^*)) \geq cost_f(S) + cost_s(S).$$

**THEOREM 2** *The locality gap of the local search algorithm where the only operation allowed is multiple swaps is* $max\{3, 5 - 2\left(\frac{p-1}{q-1}\right)\}$*, where* $q$ *is the size of the largest set* $A_i$*.*

The total number of multi-swap operations considered in Lemmas 2 and 3 and Corollaries 1 and 3 is at most $k$. The number of multi-swap operations considered in Corollary 2 is at most $k^2$. Therefore, the number of multi-swap operations considered by our analysis is at most $k^2 + k$.

### 2.3.3 Special Case When the Ratio of the Biggest Facility Cost to the Smallest Facility Cost Is Less Than $p + 1$

Add inequality (2.19) for all pairs $\langle s, o\rangle \in (A_i - b_i) \times B_i$ and then multiply by $\frac{1}{q_i - 1}$ to get

$$\sum_{o\,\in B_i} f_o - \frac{q_i}{q_i - 1}\sum_{s\,\in A_i - b_i} f_s + \sum_{j\,\in N_S^*(B_i)}(o_j - s_j) + \frac{q_i}{q_i - 1}\sum_{j\,\in N_S(A_i - b_i)} 2o_j \geq 0. \tag{2.27}$$

Since the ratio of the biggest facility cost to the smallest facility cost is less than $p + 1$ then $\sum_{o\in B_i} f_o - f_{b_i} \geq 0$. Also, since $q_i \geq p + 1$ then $\frac{p+1}{p} \geq \frac{q_i}{q_i - 1}$. Therefore, if we add the following non-negative terms $\sum_{o\in B_i} f_o - f_{b_i}$, $\frac{1}{q_i - 1}\sum_{s\in A_i - b_i} f_s$, $\frac{q_i}{q_i - 1}\sum_{j\in N_S(b_i)} 2o_j$ to (2.27) and replace $\frac{q_i}{q_i - 1}$ with $\frac{p+1}{p}$ we get

$$2\sum_{o\,\in B_i} f_o - \sum_{s\,\in A_i} f_s + \sum_{j\,\in N_{S^*}(B_i)}(o_j - s_j) + \frac{p+1}{p}\sum_{j\,\in N_S(A_i)} 2o_j \geq 0. \tag{2.28}$$

If we proceed similarly as in Section 2.3.2 and add inequalities (2.9) for all sets $A_i$, $i \in I^c$, inequalities (2.28) for all sets $A_i$, $i \in I$, and inequality (2.24), we get

$$\left[3 + \frac{2}{p}\right] (cost_f(S^*) + cost_s(S^*)) \geq cost_f(S) + cost_s(S).$$

**THEOREM 3** *The locality gap of the local search algorithm where the only operation allowed is multiple swaps for the special case when the ratio of the biggest facility cost to the smallest facility cost is less than $p+1$ is $3 + \frac{2}{p}$.*

## 2.4 Tight Example

Figure 2.3 illustrates an instance of the $k$-facility location problem showing a locality gap of 4.5 for $p = 2$ and $q = 5$, matching the locality gap stated in Theorem 1. In Figure 2.3 the pentagonal nodes form the global optimal solution $S^* = \{o_1, o_2, ..., o_k\}$, the square nodes are the local optimal solution $S = \{s_1, s_2, ..., s_k\}$, and the circular nodes are the clients. The facility costs are the integers inside the nodes and the service costs are equal to the lengths of the shortest paths between the corresponding nodes. The length of each edge is shown beside the edge and if there is no path between two nodes the distance between them is infinity.
In the instance shown in Figure 2.3, $cost(S) = \frac{18k-52}{5}$ and $cost(S^*) = \frac{4k+4}{5}$, therefore the locality gap is $\frac{18k-52}{4k+4}$ which approaches 4.5 as $k$ grows. We now prove that $S$ is locally optimal by considering all possible swaps. Let set $\{s_i, s_j\} \subset S$ be swapped with $\{o_l, o_m\} \subset S^*$.

1. If $i, j \leq u = \frac{k-4}{5}$, then $o_l$ and $o_m$ should lie in the same connected components containing $s_i$ and $s_j$, so this swap increases the cost by 4.

2. If $i \leq u < j$, then one of $o_l$, $o_m$ should lie in the same connected component as $s_i$. Without loss of generality, consider that $o_l$ lies in the same component as $s_i$. If $o_m$ lies in the same component also, the cost remains unchanged. If $m \leq k-4$ and $o_m$ lies in a different component than $s_i$ the cost increases by 2. If $m > k-4$ the cost increases by 3.5.

3. If $i, j > u$, there are three cases that need to be considered. First, if $l, m \leq k-4$ the cost remains unchanged. Second, if $l \leq k-4 < m$ then the cost increases by 1. Third, if $k-4 < l, m$ the cost increases by 2.

   The example can be generalized to arbitrary values of $p \geq 2$ and $q \geq p+3$ as shown in Figure 2.4. The local optimal solution is $S = \{s_1, s_2, ..., s_k\}$ and the global optimal solution is $S^* = \{o_1, o_2, ..., o_k\}$. The cost of $S$ is $\frac{[5(q-1)-2(p-1)]k-[(q-1)(4q-2p-3)]}{q}$ and the cost of $S^*$ is $\frac{(q-1)k+(q-1)}{q}$, so the locality gap is $\frac{5(q-1)-2(p-1)}{q-1} = 5 - 2\left(\frac{p-1}{q-1}\right)$. Note that in our tight example $k$ must be much larger than $q$.

   The proof that $S$ is locally optimal is similar as that for the case $p = 2$ and $q = 5$, but it involves many more cases.

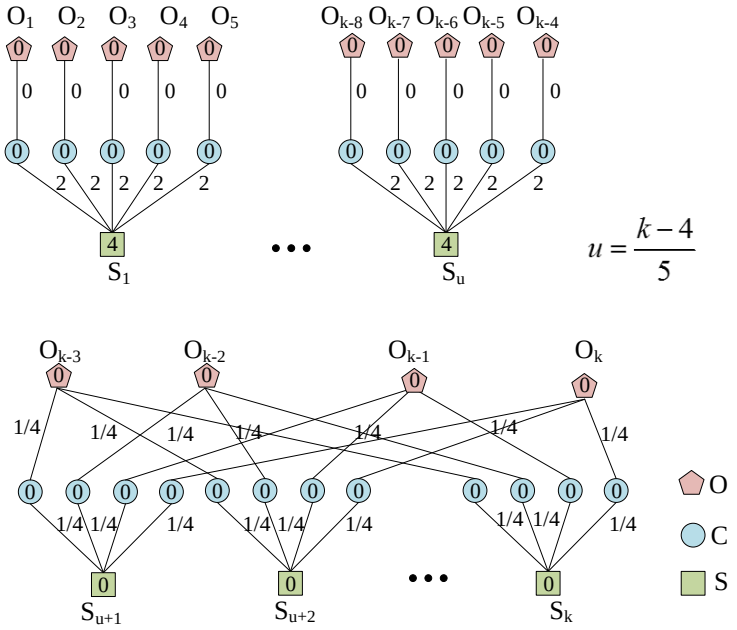


Figure 2.3: Tight example for $p = 2$ and $q = 5$.

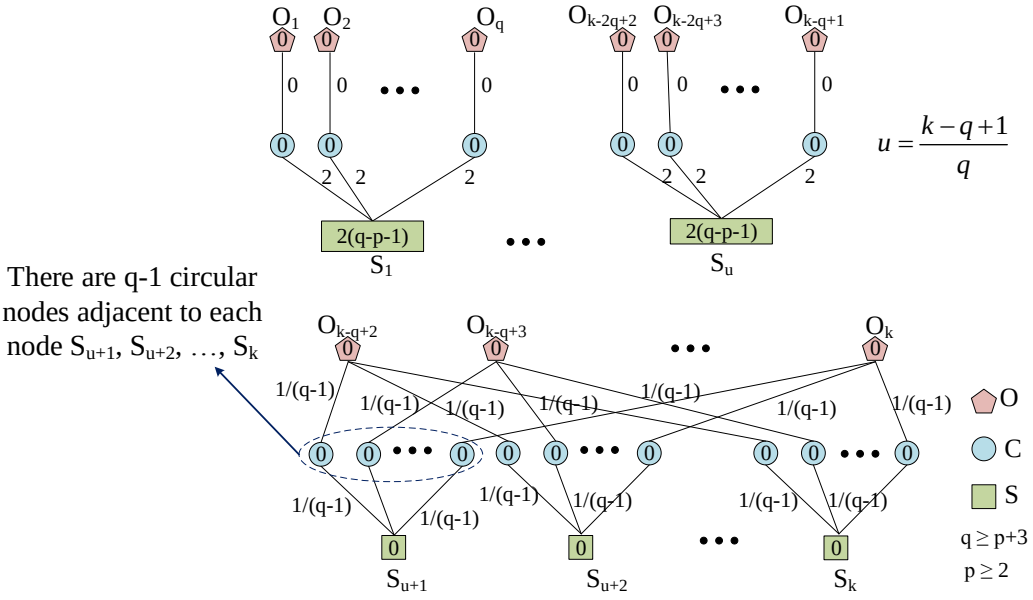


Figure 2.4: Tight example for arbitrary $p$ and $q$.

## 2.5 Scaling the Costs

We now perform a different analysis of our algorithm which yields the same bound for the locality ratio as that of the algorithm by Zhang [14] which uses facility insertions

and removals in addition to swaps. The idea is to multiply each facility cost by some value $\beta > 0$ and then compute a local optimal solution for the new problem. By carefully choosing the value of $\beta$ and some set of swap operations involving facilities from the local optimum solution and a global optimum solution we can prove a locality ratio that matches that of [14].

### 2.5.1 Bounding the Facility Cost

This time we consider the following swap operations:

1. Swap $\langle A_i, B_i \rangle$, where $|A_i| \leq p$.
2. Swap $\langle \hat{A}_i, \hat{B}_i \rangle$, where $|A_i| > p$, and swaps $\langle\langle A_i \setminus \hat{A}_i, B_i \setminus \hat{B}_i \rangle\rangle$, where $\langle\langle A_i \setminus \hat{A}_i, B_i \setminus \hat{B}_i \rangle\rangle$ denotes the set of single swaps for each pair of facilities in $A_i \setminus \hat{A}_i$ and $B_i \setminus \hat{B}_i$.
3. Swaps $\langle\langle A_r, B_r \rangle\rangle$, where $\langle\langle A_r, B_r \rangle\rangle$ denotes a set of single swaps for each pair of facilities in $A_r$ and $B_r$.

**Corollary 2.5.1** *For swap $\langle A_i, B_i \rangle$, where $|A_i| \leq p$ and $1 \leq i < r$,*

$$\sum_{o \in B_i} f_o - \sum_{s \in A_i} f_s + \sum_{j \in N_S(A_i)} 2o_j \geq 0. \tag{2.29}$$

**Proof** The corollary follows from Lemma 2. Note that as pointed out at the end of the proof of Lemma 2 we can replace in (2.9) the last term $\sum_{j \in N_S(A_i)} 2o_j$ with $\sum_{j \in N_S(A_i) \setminus N_S(B_i)} 2o_j$. In addition, since $o_j - s_j \leq 2o_j$ the third term in (2.9) can be replaced by $\sum_{N_S(B_i)} 2o_j$. The third term in (2.29) is then obtained by adding $\sum_{j \in N_S(A_i) \setminus N_S(B_i)} 2o_j$ to $\sum_{N_S(B_i)} 2o_j$.

**Corollary 2.5.2** *For swap $\langle \hat{A}_i, \hat{B}_i \rangle$, where $|A_i| > p$ and $1 \leq i < r$,*

$$\sum_{o \in \hat{B}_i} f_o - \sum_{s \in \hat{A}_i} f_s + \sum_{j \in N_S^*(\hat{B}_i)} 2o_j + \sum_{\substack{j \in N_S(\hat{A}_i) \setminus N_S*(\hat{B}_i) \\ \pi(\sigma^*(j)) \in \hat{A}_i}} (o_j + s_j) + \sum_{\substack{j \in N_S(\hat{A}_i) \setminus N_S*(\hat{B}_i) \\ \pi(\sigma^*(j)) \notin \hat{A}_i}} 2o_j \geq 0. \tag{2.30}$$

**Proof** The corollary follows from Lemma 3 by noting that $o_j - s_j \leq 2o_j$ for all clients $j$, and using this inequality in the third term of (2.10).

Lemma 4 bounds the cost increase caused by swaps $\langle s, o \rangle$, where $s \in A_i \setminus \hat{A}_i$, and $o \in B_i \setminus \hat{B}_i$. Adding (2.30) and (2.16) we get

$$\sum_{o \in B_i} f_o - \sum_{s \in A_i} f_s + \sum_{j \in N_S(A_i)} 2o_j \geq 0. \tag{2.31}$$

The third term in (2.31) is obtained from the following equalities: Adding the third term of (2.16) and the fourth term in (2.30) we get

$$\sum_{\substack{j\in N_S(\hat{A}_i)\setminus N_S^*(\hat{B}_i),\\ \pi(\sigma^*(j))\in\hat{A}_i}}(o_j - s_j) + \sum_{\substack{j\in N_S(\hat{A}_i)\setminus N_S^*(\hat{B}_i),\\ \pi(\sigma^*(j))\in\hat{A}_i}}(o_j + s_j) = \sum_{\substack{j\in N_S(\hat{A}_i)\setminus N_S^*(\hat{B}_i),\\ \pi(\sigma^*(j))\in\hat{A}_i}}2o_j.$$

Adding the right hand side of this equation and the last term in (2.30) we have

$$\sum_{\substack{j\in N_S(\hat{A}_i)\setminus N_S^*(\hat{B}_i),\\ \pi(\sigma^*(j))\in\hat{A}_i}}2o_j + \sum_{\substack{j\in N_S(\hat{A}_i)\setminus N_S^*(\hat{B}_i),\\ \pi(\sigma^*(j))\notin\hat{A}_i}}2o_j = \sum_{j\in N_S(\hat{A}_i)\setminus N_S^*(\hat{B}_i)}2o_j.$$

Finally, adding the right hand side of this equation to the third term in (2.30) and the last term in (2.16) we get

$$\sum_{j\in N_S(\hat{A}_i)\setminus N_S^*(\hat{B}_i)}2o_j + \sum_{j\in N_S^*(\hat{B}_i)}2o_j + \sum_{j\in N_S(A_i)\setminus N_S(\hat{A}_i)}2o_j = \sum_{j\in N_S(A_i)}2o_j.$$

In Corollary 2.5.1 if we replace $A_i$ with $s$ and $B_i$ with $o$, then for each swap $\langle s, o\rangle$, where $s \in A_r$ and $o \in B_r$, we have

$$f_o - f_s + \sum_{j\in N_S(s)} 2o_j \geq 0. \tag{2.32}$$

Adding all the cost increase inequalities for swaps $\langle s, o\rangle$, where $s \in A_r$ and $o \in B_r$, we get

$$\sum_{o\in B_r} f_o - \sum_{s\in A_r} f_s + \sum_{j\in N_S(A_r)} 2o_j \geq 0. \tag{2.33}$$

Adding inequalities (2.29) for all subsets $A_i$, where $|A_i| \leq p$, inequalities (2.31) for all subsets $A_i$, where $|A_i| > p$, and inequality (2.33) we get

$$\sum_{o\in S^*} f_o - \sum_{s\in S} f_s + 2\sum_{j\in C} o_j \geq 0. \tag{2.34}$$

Therefore,

$$cost_f(S) \leq cost_f(S^*) + 2cost_s(S^*). \tag{2.35}$$

### 2.5.2 Bounding the Service Cost

For bounding the service cost of the local optimum solution $S$ we consider the following swaps:

1. Swap $\langle A_i, B_i\rangle$, where $|A_i| \leq p$.
2. For each set $A_i$ with $q_i = |A_i| > p$, $1 \leq i < r$, we pair each of the $q_i - 1$ good facilities in $A_i$ with all $q_i$ facilities in $B_i$; this produces $(q_i - 1)q_i$ different pairs. For set $A_r$ we select any $q_r - 1 = |A_r| - 1$ good facilities in $A_r$ and we pair each one of them with all facilities in $B_r$. For each $i = 1, 2, ..., r$, we swap each one of the $(q_i - 1)q_i$ pairs of facilities.

By Lemma 2, for each swap $\langle A_i, B_i\rangle$, where $|A_i| \leq p$,

$$\sum_{o \in B_i} f_o - \sum_{s \in A_i} f_s + \sum_{j \in N_{S^*}(B_i)} (o_j - s_j) + \sum_{j \in N_S(A_i)} 2o_j \geq 0. \tag{2.36}$$

By Lemma 2.3.2, for each swap $\langle s, o\rangle$, where $s \in A_i$ is a good facility, $o \in B_i$, $1 \leq i < r$ and $|A_i| > p$,

$$f_o - f_s + \sum_{j \in N^*_S(o)} (o_j - s_j) + \sum_{j \in N_S(s)} 2o_j \geq 0. \tag{2.37}$$

Adding all the cost increase inequalities related to swapping all $q_i(q_i - 1)$ pairs $\langle s, o\rangle$, where $s \in A_i$ is a good facility and $o \in B_i$, and then multiplying by $\frac{1}{q_i - 1}$ we get

$$\sum_{o \in B_i} f_o - \frac{q_i}{q_i - 1} \sum_{s \in A_i \setminus b_i} f_s + \sum_{j \in N_{S^*}(B_i)} (o_j - s_j) + \frac{q_i}{q_i - 1} \sum_{j \in N_S(A_i)} 2o_j \geq 0. \tag{2.38}$$

where $b_i \in A_i$ is a bad facility.

Since $\frac{q_i}{q_i - 1} \leq \frac{p+1}{p}$ and $2o_j \geq 0$, inequality (2.38) can be rewritten as

$$\sum_{o \in B_i} f_o - \frac{q_i}{q_i - 1} \sum_{s \in A_i \setminus b_i} f_s + \sum_{j \in N_{S^*}(B_i)} (o_j - s_j) + \frac{p+1}{p} \sum_{j \in N_S(A_i)} 2o_j \geq 0. \tag{2.39}$$

Let $I = \{i \mid 1 \leq i \leq r, |A_i| > p\}$ and $I^c = \{i \mid 1 \leq i \leq r, |A_i| \leq p\}$. Adding inequalities (2.36) for all sets $A_i$, where $i \in I^c$, and inequalities (2.39) for all $A_i$, where $i \in I$, we get

$$\sum_{o \in S^*} f_o - \sum_{i \in I^c} \sum_{s \in A_i} f_s - \sum_{i \in I} \left[ \left( \frac{q_i}{q_i - 1} \right) \sum_{s \in A_i \setminus b_i} f_s \right] + \sum_{j \in C} o_j - \sum_{j \in C} s_j + \left( \frac{p+1}{p} \right) \sum_{j \in C} 2o_j \geq 0. \tag{2.40}$$

The sixth term is obtained by noting that $\sum_{i \in I^c} \left( \frac{p+1}{p} \right) \sum_{j \in N_S(A_i)} 2o_j \geq \sum_{i \in I^c} \sum_{j \in N_S(A_i)} 2o_j$.

Therefore,

$$cost_f(S^*) - \sum_{i \in I^c} \sum_{s \in A_i} f_s - \sum_{i \in I} \left[ \left( \frac{q_i}{q_i - 1} \right) \sum_{s \in A_i \setminus b_i} f_s \right] + cost_s(S^*) - cost_s(S) + \left( \frac{p+1}{p} \right) 2cost_s(S^*) \geq 0$$

$$\Rightarrow \sum_{i \in I^c} \sum_{s \in A_i} f_s + \sum_{i \in I} \left[ \left( \frac{q_i}{q_i - 1} \right) \sum_{s \in A_i \setminus b_i} f_s \right] + cost_s(S) \leq cost_f(S^*) + \left( 3 + \frac{2}{p} \right) cost_s(S^*). \tag{2.41}$$

Since the term $\sum_{i \in I^c} \sum_{s \in A_i} f_s + \sum_{i \in I} \left[ \left( \frac{q_i}{q_i - 1} \right) \sum_{s \in A_i \setminus b_i} f_s \right]$ is positive, we can omit it from inequality (2.41) and we get

$$cost_s(S) \leq cost_f(S^*) + \left[ 3 + \frac{2}{p} \right] cost_s(S^*). \tag{2.42}$$

**THEOREM 4** *Algorithm Local Search has locality gap* $2 + \frac{1}{p} + \sqrt{3 + \frac{2}{p} + \frac{1}{p^2}}$.

**Proof** Consider an instance $(F, C)$ of the $k$-facility location problem. Multiply the cost of each facility in $F$ by some value $\beta > 0$ and compute a local optimum solution $S$ for the new instance. Let $cost'_f(X)$ and $cost'_s(X)$ denote respectively the facility and service cost of solution $X$ in the new scaled problem, and let $cost_f(X)$ and $cost_s(X)$ be the facility and service cost of solution $X$ with the original costs. Using inequalities (2.35) and (2.42) we get

$$cost'_f(S) \leq cost'_f(S^*) + 2cost'_s(S^*)$$

and

$$cost'_s(S) \leq cost'_f(S^*) + \left(3 + \frac{2}{p}\right) cost'_s(S^*).$$

Since

$$cost_f(S) + cost_s(S) = \frac{cost'_f(S)}{\beta} + cost'_s(S)$$

then

$$cost_f(S) + cost_s(S) \leq \frac{cost'_f(S^*) + 2cost'_s(S^*)}{\beta} + cost'_f(S^*) + \left(3 + \frac{2}{p}\right) cost'_s(S^*)$$

$$= \left(1 + \frac{1}{\beta}\right) cost'_f(S^*) + \left(3 + \frac{2}{p} + \frac{2}{\beta}\right) cost'_s(S^*)$$

$$= (\beta + 1) cost_f(S^*) + \left(3 + \frac{2}{p} + \frac{2}{\beta}\right) cost_s(S^*).$$

By setting $\beta = 1 + \frac{1}{p} + \sqrt{3 + \frac{2}{p} + \frac{1}{p^2}}$ we get

$$cost(S) \leq \left[2 + \frac{1}{p} + \sqrt{3 + \frac{2}{p} + \frac{1}{p^2}}\right] cost(S^*).$$

The total number of multi-swap operations considered in Sections 2.5.1 and 2.5.2 is at most $k^2 + k$.

# Bibliography


[1] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, V. Pandit, Local search heuristic for k-median and facility location problem, SIAM Journal on Computing 33, (2004), 544-562.

[2] N. Atay and B. Bayazit. Mobile Wireless Sensor Network Connectivity Repair with K-Redundancy. Algorithmic Foundations of Robotics. Editors G.S. Chirikjian, H. Choset, M. Morales, and T. Murphey, Springer Tracts in Advanced Robotics, (2010) 35-52.

[3] M. Charikar , S. Guha, Improved Combinatorial Algorithms for Facility Location Problems, SIAM Journal on Computing 34, (2005), 803-824.

[4] M. Charikar , S. Guha, Improved Combinatorial Algorithms for the Facility Location and $k$-median Problems, Proceeding of the 40th IEEE Annual Symposium on Foundations of Computer Science, (1999), 378-388.

[5] M. Charikar , S. Guha, E. Tardos, and D. Shmoys, A constant-factor approximation algorithm for the k-median problem. Journal of Computer and System Science 65, (2002), 129-149.

[6] G. Cornuejols, M. L. Fisher, and G. L Nemhauser, Exceptional paper-location of bank accounts to optimize float: An analytic study of exact and approximate algorithm. Management Science, 23, (1977), 789-810.

[7] A. Gupta, K. Tangwongsan, Simpler Analyses of Local Search Algorithms for Facility Location. CoRR, abs/0809.2554, 2008.

[8] K. Jain, V. Vazirani, Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation, Journal of the ACM, 48 (2001), 274-296.

[9] K. Jain, M. Mahdian, E. Markakis, A. Saberi, V. Vazirani, Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP, Journal of the ACM, 50, (2003), 795-824.

[10] A. Klose, A. Drexl, Facility location models for distribution system design, European Journal of Operational Research 162 (2005) 4–29.

[11] S. Li, A 1.488 approximation algorithm for the uncapacitated facility location problem. Information and Computation, 222, (2013), 45-58.

[12] L. Qiu, V. Padmanabhan, G. Voelker, On the placement of web server replicas, in: Proceedings of IEEE INFOCOM'01, 2001, pp. 1587-1596.

[13] D. Shmoys, E. Tardos, and K. Aardal, Approximation algorithms for facility location problems, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, ACM, New York, 1997, pp. 265-274.

[14] P. Zhang, A new approximation algorithm for the k-facility location problem. Theoretical Computer Science, 384, (2007), 126-135.

# Chapter 3

# A Local Search Algorithm for the Multiway Cut Problem

## 3.1 Introduction

Given an undirected graph $G = (V, E)$ with non-negative weights or costs on the edges and a set $T = \{t_1, t_2, ..., t_k\} \subseteq V$ of $k$ terminals, a multiway cut is a set $E' \subseteq E$ of edges whose removal from $G$ separates all the terminals from each other and thus partitions the vertices into $k$ disjoint sets none of which contains two terminals. In the multiway cut problem the goal is to find a multiway cut with minimum weight. If the set of terminals is not given and the goal is to find a minimum weight set of edges whose removal partitions the graph into $k$ non-empty components the problem is known as the minimum $k$-cut problem. For the case when $k = 2$ the multiway cut problem reduces to the well known minimum s-t cut problem and the minimum $k$-cut problem reduces to the minimum cut problem. For $k \geq 3$ Dahlhaus et al [8] proved that the multiway cut problem is MAX SNP-hard. Goldschmidt and Hochbaum [9] showed that the minimum $k$-cut problem is **NP**-hard for arbitrary $k$.

The multiway cut problem has a variety of applications including task scheduling in multi-processors systems [17], task allocation in parallel computing systems [12], labelling the pixels of an image [3], and integrated circuit layout design [1, 13].

Dahlhaus et al. [8] presented the first approximation algorithm for the multiway cut problem. They used a simple combinatorial technique, called the isolation heuristic, to yield an approximation algorithm with approximation ratio $2 - \frac{2}{k}$. In the isolation heuristic a minimum weight cut is found that separates terminal $t_i$ from $T - \{t_i\}$, for each $1 \leq i \leq k$; then the solution is formed by the union of the $k-1$ smallest cuts. Calinescu et al [6] utilized an elegant geometric relaxation algorithm and improved the approximation ratio to $1.5 - \frac{1}{k}$. Karger et al. [10] used a similar geometric relaxation technique to design a $\frac{12}{11}$-approximation algorithm for the problem for the case when $k = 3$ and for general $k$ they improved the approximation ratio to 1.3438. Independently, Cunningham and Tang [7] designed an approximation algorithm with the same approximation ratio $\frac{12}{11}$ for the case $k = 3$. Sharma and Vondrak [16] designed a better algorithm with approximation ratio 1.2965 based on the linear programming relaxation used in [6]. Buchbinder et al.

[4, 5] matched Sharma and Vondrak's approximation ratio, but using a much simpler algorithm. These last two algorithms have the currently best known approximation ratio for the multiway cut problem.

Naor and Zosin [14] gave a 2-approximation algorithm for the version of the multiway cut problem on directed graphs. Zhao [18] et al, used a greedy splitting technique to design an approximation algorithm with ratio $2 - \frac{2}{k}$ for both the multiway cut and the minimum $k$-cut problem. Saran and Vazirani [15] used Gromory-Hu cuts to design an approximation algorithm with the same $2 - \frac{2}{k}$ approximation ratio for the minimum $k$-cut problem. Goldschmidt and Hochbaum [9] gave a polynomial time algorithm for the minimum $k$-cut problem when the value of $k$ is fixed.

In this paper we present a local search algorithm for the multiway cut problem with approximation ratio $2 - \frac{2}{k}$. Our algorithm utilizes the expansion move introduced in [3]. We also give an example showing the tightness of our analysis. In addition, we show that a slight modification of our local search algorithm can be used on two variations of the multiway cut problem: 1) when some set of nodes needs to be in the same partition, and 2) when certain nodes can only be in some partitions. This paper also includes an experimental comparison of our local search algorithm with four other approximation algorithms for the multiway cut problem: Dahlhaus et al. [8] isolation heuristic, the algorithm of Calinescu et al. [6], the algorithm of Sharma and Vondrak [16], and the algorithm of Buchbinder et al. [4, 5].

Even though our algorithm does not achieve the same approximation ratio as the best known approximation algorithm for the multiway cut problem, we consider our work of interest as there are not many local search approximation algorithms with proven performance guarantee. Our algorithm is simple and intuitive and its analysis, even though it seems very complicated, it is based on a natural idea of comparing the weight of a global optimum solution and a local optimum one based on the fact that no local change on a local optimum solution can improve its value.

In addition, according to our experimental results our local search algorithm performs much better than the isolation heuristics algorithm even though they have the same theoretical worst-case approximation ratio. Furthermore, has comparable performance to the three currently best known algorithms for the multiway cut problem: the algorithm of Calinescu et al. [6], the algorithm of Sharma and Vondrak [16], and the algorithm of Buchbinder et al. [4, 5].

## 3.2 The Local Search Algorithm

As mentioned in the previous section our local search algorithm uses the expansion moves of [3] to find a local optimal solution. Here for convenience we rename the expansion move and call it the relabel operation. Given a graph $G = (V, E)$ and a set $L$ of labels, a labelling function $f$ assigns a label to each node of $G$. A relabel operation is determined by a set of nodes $A \subseteq V \setminus T$, a label $\alpha$ and a labelling function $f$. A relabel operation modifies the labelling function $f$ by changing the labels of all the nodes in $A \subseteq V \setminus T$ to $\alpha$ while keeping the other labels unchanged, and is defined as follows:

$$R\langle A, \alpha, f\rangle := \forall u \in A, f(u) = \alpha.$$

We can formulate the multiway cut problem as a labelling problem as follows: Given a graph $G = (V, E)$, a set of $k$ terminals $T = \{t_1, t_2, ..., t_k\} \subseteq V$ and a set of $k$ labels $L = \{\alpha_1, \alpha_2, ..., \alpha_k\}$, where each terminal $t_i$ has a fixed label $\alpha_i$, the goal is to label the nodes in $V - T$ in such a way as to minimize the total cost of the set of edges $E'$ whose endpoints have different labels. Note that $E'$ is a multiway cut for $G$ and therefore finding a solution for the above labelling problem is equivalent to finding a minimum weight multiway cut. The cost of a labelling function $f$ is defined as the total cost of the edges whose endpoints have different labels.

The *neighborhood function* for a given labelling function $f$ and label $\alpha_i$, $i = 1, 2, ..., k$ is defined as follows:
$\mathcal{N}_i(f) =$
$\{$all the labellings $f'$ that can be obtained by $R\langle A, \alpha_i, f\rangle$, for all possible sets $A \subseteq V \setminus T\}$.

Our local search algorithm for the multiway cut problem is described below.

**Algorithm 4** MULTIWAY CUT ($G = (V, E)$, $L$, $T$)

```
1: Input: Graph G = (V, E), set L of k labels, and set T ⊆ V of k terminals
2: Output: Labelling f of a local optimum solution for the multiway cut problem
3: f ← any arbitrary labelling function that assigns to each terminal a different label
4: success ← 1
5: while success = 1 do
6:     success ← 0
7:     for i ← 1 to k do
8:         Compute a minimum cost labelling f' ∈ N_i(f)
9:         if cost of the solution defined by labelling f' is less than the cost of the solution
   defined by labelling f then
10:            f ← f'
11:            success ← 1
12:        end if
13:    end for
14: end while
15: return f
```

Note that algorithm MULTIWAY CUT might not run in time that is polynomial in the size of the input as every iteration of the while loop might only provide a marginal improvement in the cost of the solution, so it might require a very large number of iterations to find a local optimum solution. We can proceed as in [2] to ensure a polynomial running time: Replace the condition of the if statement as follows "If cost of the solution defined by labelling $f'$ is less than $(1 - \epsilon)$ times the cost of the solution defined by labelling $f$", where $\epsilon$ is a positive value.

With this change the maximum number of iterations of the outer loop is $O(\frac{(\log n + log(c_{max}))}{\epsilon})$, where $n$ is the number of vertices and $c_{max}$ is the largest edge cost. Each iteration of the loop needs polynomial time, so the running time of the algorithm is polynomial. This change causes the cost of the solution produced by the algorithm to be at most a factor $\frac{1}{1-\epsilon}(2 - \frac{2}{k})$ away from the optimum.

## 3.3 Finding a minimum cost relabel operation

In Line 9 of algorithm Local Search we need to compute a minimum cost labelling $f'$ obtainable from the given labelling $f$ through one relabel operation. In this section we present an algorithm based on an algorithm in [3] to find this optimal labelling $f'$. Given a weighted graph $G' = (V', E')$ and two distinguished nodes called *source* and *sink*, a *cut* is a set of edges $C \subseteq E$ that separates *source* from *sink*. A cut $C$ is *minimal* if no subset of $C$ is a cut. The *cost of a cut* $C$ is the sum of the costs of the edges in $C$. We call $C$ a *minimum cut* if it is a minimal cut with minimum cost.

We now define a graph $G_\alpha = (V_\alpha, E_\alpha)$, where $\alpha$ is a label, which has the property that a minimum cut of $G_\alpha$ determines a relabel operation $R\langle A, \alpha, f\rangle$ that produces a minimum cost labelling $f'$ as required by the algorithm.

The set of nodes of $G_\alpha$ includes a source node $\alpha$, a sink node $\overline{\alpha}$, all nodes $V$ of the input graph $G$ and a set of *auxiliary* nodes $\bigcup_{\substack{\{p,q\}\in V,\\ (p,q)\in E}} a_{\{p,q\}}$. Formally,

$$V_\alpha = \{\{\alpha, \overline{\alpha}\}, V, \{\bigcup_{\substack{\{p,q\}\subseteq V,\\ (p,q)\in E}} a_{\{p,q\}}\}\} \tag{3.1}$$

As for the set of edges of $G_\alpha$, all nodes in $V$ are adjacent to source $\alpha$ and sink $\overline{\alpha}$. Also nodes $p$, $q$ that are adjacent in $G$ and have the same label, i.e. $f(p) = f(q)$, are adjacent in $G_\alpha$. Finally, for each pair $(p, q)$ of adjacent nodes in $G$ with $f(p) \neq f(q)$ there is a triplet of edges, $\varepsilon_{\{p,q\}} = \{(p, a), (a, q), (a, \overline{\alpha})\}$, where $a = a_{\{p,q\}}$ is an auxiliary node, so both nodes $p$ and $q$ are connected to auxiliary node $a$ and $a$ is connected to the sink $\overline{\alpha}$. Therefore, the set of edges in $G_\alpha$ is,

$$E_\alpha = \bigcup_{p\in V}\{(p,\alpha),(p,\overline{\alpha})\} \bigcup_{\substack{\{p,q\}\subseteq V\\ (p,q)\in E\\ f(p)\neq f(q)}} \varepsilon_{\{p,q\}} \bigcup_{\substack{\{p,q\}\subseteq V\\ (p,q)\in E\\ f(p)=f(q)}} (p,q) \tag{3.2}$$

The edges have assigned weights as shown in Table 3.1, where $P_\alpha$ is the set of nodes labelled $\alpha$ in $G$. Figure 3.1 shows an example of graphs $G$ and $G_\alpha$.

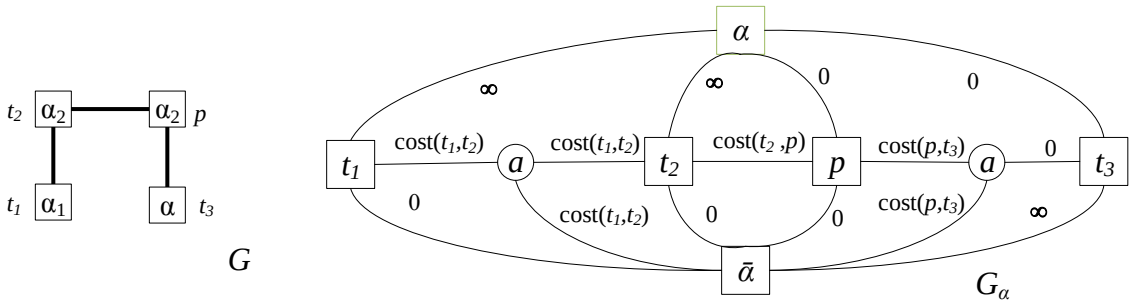


Figure 3.1: In graph $G_\alpha$ nodes represented by squares are the nodes in $G$ and nodes labelled with $t_i$ where $i = \{1, 2, 3\}$ are terminals in $G$. Auxiliary nodes are represented by circles. In $G$ labels assigned to the nodes are inside the squares.

| Edge | Weight | Condition |
|---|---|---|
| $(\alpha, t_i)$ | $\infty$ if $t_i \notin P_\alpha$<br>0 if $t_i \in P_\alpha$ | $t_i \in T$ |
| $(\overline{\alpha}, t_i)$ | 0 if $t_i \notin P_\alpha$<br>$\infty$ if $t_i \in P_\alpha$ | $t_i \in T$ |
| $(\alpha, p)$ | 0 | $p \in V - T$ |
| $(\overline{\alpha}, p)$ | 0 | $p \notin P_\alpha$ |
| $(\overline{\alpha}, p)$ | $\infty$ | $p \in P_\alpha$ |
| $(p, q)$ | $cost(p, q)$ | $f(p) = f(q)$ |
| $(p, a)$ | 0 if $p \in P_\alpha$<br>$cost(p, q)$ if $p \notin P_\alpha$ | $f(p) \neq f(q)$ |
| $(a, q)$ | 0 if $q \in P_\alpha$<br>$cost(p, q)$ if $q \notin P_\alpha$ | $f(p) \neq f(q)$ |
| $(\overline{\alpha}, a)$ | $cost(p, q)$ | $f(p) \neq f(q)$ |

Table 3.1: Weights for the edges in $G_\alpha$.

**Lemma 6** *Let $C$ be a minimal cut of $G_\alpha$ separating $\alpha$ from $\bar{\alpha}$. For each vertex $p \in V$ exactly one the edges $(p, \alpha)$, $(p, \overline{\alpha})$ belongs to $C$.*

**Proof** Let $G_C = (V_\alpha, E_\alpha \setminus C)$ be the graph obtained by removing from $G_\alpha$ the edges in $C$. At least one of $(p, \alpha)$, $(p, \overline{\alpha})$ must belong to $C$ as otherwise $\alpha$, $p$, $\overline{\alpha}$ would be a path in $G_c$.

Assume that $(p, \alpha) \in C$ and $(p, \overline{\alpha}) \in C$. Since $C$ is a minimal cut then if we remove $(p, \alpha)$ from $C$ there must be at least one path $P = \overline{\alpha}, p_1, p_2, ..., p_i, p, \alpha$ in $G_{C \setminus \{(p,\alpha)\}}$. Similarly, if we remove $(p, \overline{\alpha})$ from $C$ there must be at least one path $P' = \overline{\alpha} p, p'_1, p'_2, ..., p'_j, \alpha$ in $G_{C \setminus \{(p,\overline{\alpha})\}}$ (see Figure 3.2). However, then $\overline{\alpha}, p_1, p_2, ..., p_i, p, p'_1, p'_2, ..., p'_j, \alpha$ is a path in $G_C$ contradicting the fact that $C$ is a cut of $G_\alpha$ separating $\alpha$ from $\bar{\alpha}$.

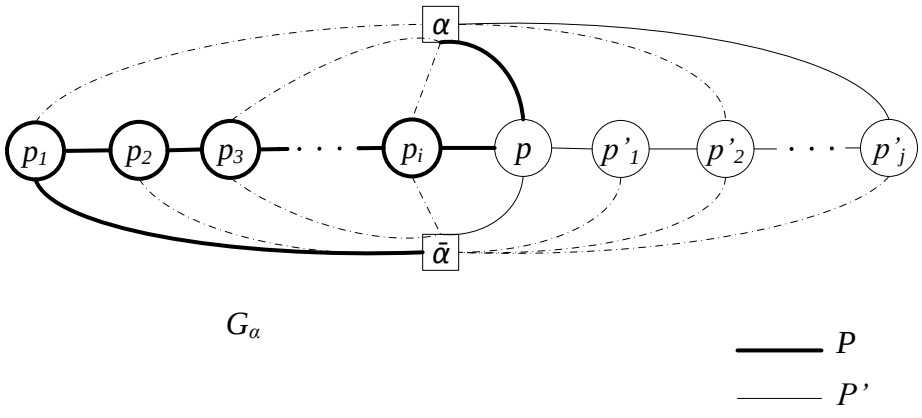


Figure 3.2: Paths $P$ and $P'$ are shown above in thick and thin solid lines, respectively.

**Lemma 7** *Each minimal cut $C$ of $G_\alpha$ of bounded cost separating $\alpha$ from $\bar{\alpha}$ defines a labelling $f'$ for $G$ that can be obtained from $f$ through a relabel operation and its cost, $\sum_{\substack{(p,q)\in E \\ f'(p)\neq f'(q)}} cost(p,q)$, is at most the cost of $C$. This labelling $f'$ is defined as follows:*

$$f'(p) = \begin{cases} \alpha & if\,(p,\alpha) \in C \\ f(p) & if\,(p,\overline{\alpha}) \in C \end{cases} \quad \forall p \in V. \tag{3.3}$$

**Proof** By Lemma 6, $f'$ assigns a unique label to each node of $G$, so $f'$ is a valid labelling for $G$.

Note that if edge $(p,q) \in E$ is such that $f(p) = f(q)$ but $f'(p) \neq f'(q)$ then either,

- $(p,\alpha) \in C$ and $(q,\bar{\alpha}) \in C$ this means that $(p,q) \in C$ as otherwise $(\alpha,q)$, $(q,p)$, $(p,\bar{\alpha})$ would be a path in $G_c = (V_\alpha, E_\alpha \setminus C)$, or
- $(p,\bar{\alpha}) \in C$ and $(q,\alpha) \in C$: this means that $(p,q) \in C$ as otherwise $(\alpha,q)$, $(q,p)$, $(p,\bar{\alpha})$ would be a path in $G_c$.

On the other hand, if edge $(p,q) \in E$ is such that $f(p) \neq f(q)$ and $f'(p) \neq f'(q)$ then there are two cases that we need to consider. Let $a = a_{\{p,q\}}$.

1. $(p,\overline{\alpha}), (q,\overline{\alpha}) \in C$. By Lemma 6 none of the edges $(p,\alpha)$, $(q,\alpha)$ belong to $C$. Hence, either $(p,a)$ and $(q,a)$ must belong to $C$ or $(a,\overline{\alpha})$ must belong to $C$ as otherwise there would be a path from $\alpha$ to $\overline{\alpha}$ going through $a$. In both cases the cost of these edges is at least $cost(p,q)$.

2. $(p,\alpha), (q,\overline{\alpha}) \in C$ (the case $(q,\alpha), (p,\overline{\alpha}) \in C$ is similar). By Lemma 6, $(\alpha,q) \notin C$ and $(p,\overline{\alpha}) \notin C$. Hence, either

   - $(a,q) \in C$. Note that the cost of this edge is $cost(p,q)$: $q$ cannot have label $\alpha$ in $f$ as otherwise $cost(q,\overline{\alpha}) = \infty$.
   - $(a,\overline{\alpha}) \in C$ and $(p,a) \in C$ . Note that $cost(a,\overline{\alpha}) = cost(p,q)$.

   In both cases $cost(C \cap \varepsilon_{\{p,q\}}) \geq cost(p,q)$.

Therefore,

$$cost(C) \geq \sum_{\substack{(p,q)\in E, f(p)=f(q) \\ f'(p)\neq f'(q)}} cost(p,q) + \sum_{\substack{(p,q)\in E, f(p)\neq f(q) \\ f'(p)\neq f'(q)}} cost(p,q) = cost(f') \tag{3.4}$$

**Lemma 8** *A labelling $f'$ obtained from $f$ by a relabel operation $R(A,\alpha,f)$ defines a minimal cut $C'$ of $G_\alpha$ separating $\alpha$ from $\bar{\alpha}$ of cost equal to the cost of $f'$.*

**Proof** The cut $C'$ is defined as follows:

I) For each node $p \in V$, if $f'(p) = \alpha$ then $(p,\alpha) \in C'$ otherwise $(p,\overline{\alpha}) \in C'$.

II) For each edge $(p,q) \in E$ such that $f(p) = f(q)$ and $f'(p) \neq f'(q)$, edge $(p,q) \in C'$.

III) For each edge $(p,q) \in E$ such that $f(p) \neq f(q)$ and $f'(p) \neq f'(q)$, let $a = a_{\{p,q\}}$.

a) if $(p,\overline{\alpha}) \in C'$ and $(q,\overline{\alpha}) \in C'$ then $(a,\overline{\alpha}) \in C'$. Note that $cost(a,\overline{\alpha}) = cost(p,q)$.

b) if $(p,\alpha) \in C'$ and $(q,\overline{\alpha}) \in C'$ then $(a,q) \in C'$. Note that $f(q) \neq \alpha$ as otherwise edge $(q,\alpha)$ would belong to $C'$. Hence, $cost(a,q) = cost(p,q)$.

c) if $(p,\overline{\alpha}) \in C'$ and $(q,\alpha) \in C'$ then $(a,p) \in C'$. Similarly as above, $cost(a,p) = cost(p,q)$.

To prove that $C'$ is a cut we show that there is no path in $G_{C'}$ from $\alpha$ to $\overline{\alpha}$. The proof is by contradiction. Assume that $P = \alpha p_0 p_1 p_2 ....p_i\overline{\alpha}$ is a shortest path in $G_{C'}$ from $\alpha$ to $\overline{\alpha}$. Node $p_0$ cannot be an auxiliary node because it is connected to $\alpha$. Consider the following two cases for node $p_1$: (1) If $p_1$ is not an auxiliary node then by (I), we know $f'(p_0) \neq \alpha$ since $(p_0,\alpha) \notin C'$ and by (II) we know $f'(p_0) = f'(p_1) \neq \alpha$ since $(p_0,p_1) \notin C'$. Using (I) again then $(p_1,\overline{\alpha}) \in C'$ and so $(p_1,\alpha) \notin C'$. However then $\alpha p_1 p_2 ....p_i\overline{\alpha}$ is a path in $G_{C'}$ contradicting the assumption that $P$ was a shortest path in $G'_C$ between $\alpha$ and $\overline{\alpha}$. (2) If $p_1$ is an auxiliary node, since $(p_0,\overline{\alpha}) \in C'$ and $(p_1,p_2),(p_1,p_0) \notin C'$ then only case (IIIa) is applicable to the set of nodes $\{p_0,p_1,p_2\}$ and therefore $(p_2,\overline{\alpha}) \in C'$. However then $\alpha p_2 ....p_i\overline{\alpha}$ is a path in $G_{C'}$ contradicting the assumption that $P$ was a shortest path in $G_{C'}$ between $\alpha$ and $\bar{\alpha}$.

Because of the way in which $C'$ has been defined, it is easy to verify that $cost(f') = cost(C')$ and that $C'$ is a minimal cut because if any edge is removed from $C'$ the remaining set of edges would not form a cut of $G_\alpha$ separating $\alpha$ from $\bar{\alpha}$.

**THEOREM 5** *There is a polynomial time algorithm that given a labelling function $f$ for a graph $G = (V,E)$ and a label $\alpha$ it computes a minimum cost labelling $f'$ that can be obtained from $f$ through a single relabel operation.*

**Proof** The algorithm builds the graph $G_\alpha$ as described above, computes a minimum cut $C'$ of $G_\alpha$ separating $\alpha$ from $\bar{\alpha}$ and outputs the labelling $f'$ defined by $C'$ as described in Lemma 7. To see that $f'$ is a minimum cost labelling obtained from $f$ through a relabel operation, assume that there is a labelling $f''$ obtained from $f$ through a relabel operation and that $cost(f'') < cost(f')$. By Lemma 8, $f''$ defines a cut $C''$ of $G_\alpha$ separating $\alpha$ from $\bar{\alpha}$ and $cost(C'') = cost(f'')$. But then by Lemma 7, $cost(C'') = cost(f'') < cost(f') \leq cost(C')$ contradicting the assumption that $C'$ is a minimum cut.

## 3.4 Analysis of algorithm MULTIWAY CUT for the 3-way Cut Problem

Since the analysis of our algorithm is a bit complex, we first analyze it for the case when $k = 3$ and in the next section we consider the case when $k > 3$.

Let $\hat{f}$ be the labelling function computed by our local search algorithm and let $f^*$ be the labelling function corresponding to a global optimal solution. Note that $\hat{f}$ and $f^*$ are functions that assign to each node a label determining to which partition the node

belongs. Let $\alpha_1$, $\alpha_2$ and $\alpha_3$ be the three labels. We define partitions $\hat{A}_1$, $\hat{A}_2$, $\hat{A}_3$ in the local optimal solution and $A_1^*$, $A_2^*$, $A_3^*$ in the global optima solution as follows (see Figure 3.3):

$$\hat{A}_i = \{v \in V | \hat{f}(v) = \alpha_i\}$$

$$A_i^* = \{v \in V | f^*(v) = \alpha_i\}$$

for all $1 \leq i \leq 3$.

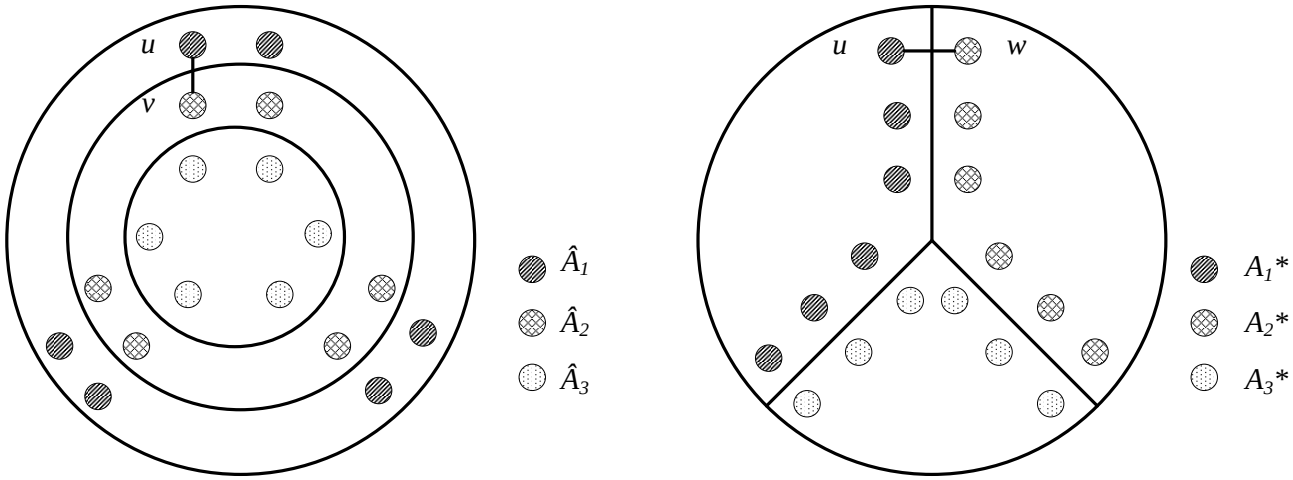


Figure 3.3: The left figure shows partitions $\hat{A}_1$, $\hat{A}_2$ and $\hat{A}_3$ of the local optimal solution and the right figure shows partitions $A_1^*$, $A_2^*$ and $A_3^*$ of the global optimal solution. In both figures the set of edges that contribute to the 3-way cut connect nodes in different partitions. For example, edge $(u, v)$ in the left figure connects $u \in \hat{A}_1$ to $v \in \hat{A}_2$ so it contributes to the cost of the local optimal solution, and edge $(u, w)$ in the right figure connects $u \in A_1^*$ to $w \in A_2^*$ so it contributes to the cost of the global optimal solution.

Our goal is to find an upper bound for the cost of the local optimal solution in terms of the cost of the global optimal solution. In order to find this upper bound we perform several relabel operations. Let $A$ be a set of nodes, $\alpha$ a label and $f$ be a labelling function. By performing a relabel operation we modify a given multi-way cut. If we perform a relabel operation on the local optimum solution, we produce a new solution of cost larger than or equal to the cost of the local optimal solution. We show how to bound the cost of the local optimal solution using this local optimality property.

Let $\hat{S}$ and $S^*$ be the sets of edges crossing the partitions of the local optimal solution and of the global optimal solution, respectively. Let $\hat{S}_p = \hat{S} - S^*$ and $S_p^* = S^* - \hat{S}$.

We classify the edges in $\hat{S}$ into three groups as follows:

$$\hat{B}_{\alpha_i\alpha_j} = \{(v, u) \in E | \hat{f}(v) = \alpha_i, \hat{f}(u) = \alpha_j\}$$

for all $1 \leq i < j \leq 3$. Similarly, the edges in $S^*$ are partitioned into groups as follows:

$$B^*{}_{\alpha_i\alpha_j} = \{(v, u) \in E | f^*(v) = \alpha_i, f^*(u) = \alpha_j\}$$

for all $1 \leq i < j \leq 3$.

Let $P$ and $Q$ be sets of nodes and $B$ be a set of edges. The sets of edges $(B|P)$ and $(B|P:Q)$ are defined as follows:

$$(B|P) = \{(u,v) \in B | u,v \in P\}$$

$$(B|P:Q) = \{(u,v) \in B |\ u \in P, v \in Q\}$$

Finally the cost $C(B)$ of set $B$ is defined as:

$$C(B) = \sum_{(u,v)\in B} cost(u,v)$$

In the next sections we compute two different bounds for the value of the local optimum solution; then we combine the bounds to compute the approximation ratio of our algorithm.

### 3.4.1 First bound

To determine the cost of the local optimal solution we need to carefully choose a set of relabel operations. We define the following sets of nodes that will participate in the relabel operations: $A_{is} = A_i^* \cap \hat{A}_i$ and ${A_{is}}^c = \hat{A}_i - A_{is}$ for $i = 1, 2, 3$.

We first perform the relabel operation $R\langle {A_{1s}}^c, \alpha_2, \hat{f}\rangle$ and determine how this changes the cost of the local optimal solution. Observe that this operation changes the label that $\hat{f}$ assign to each node in ${A_{1s}}^c$ to $\alpha_2$: thus, (1) it decreases the contribution to the cost of the solution made by the set $\Delta_1$ of edges with one endpoint in ${A_{1s}}^c$ and the other one in $\hat{A}_2$, and (2) it increases the contribution to the cost of the solution made by the set $\Delta_2$ of edges with one endpoint in ${A_{1s}}^c$ and the other in $A_{1s}$.

Note that ${A_{1s}}^c = \hat{A}_1 - A_{1s} = \hat{A}_1 - (A_1^* \cap \hat{A}_1) = \hat{A}_1 - A_1^* = \hat{A}_1 \cap (A_2^* \cup A_3^*)$ and so $\Delta_1 = (\hat{B}_{\alpha_1\alpha_2}|\hat{A}_1 \cap (A_2^* \cup A_3^*) : \hat{A}_2) = (\hat{B}_{\alpha_1\alpha_2}|\hat{A}_1 \cap (A_2^* \cup A_3^*) : \hat{A}_2 \cap V) = (\hat{B}_{\alpha_1\alpha_2}|A_2^* \cup A_3^* : V)$ as for every edge in $\hat{B}_{\alpha_1\alpha_2}$, one endpoint is in $\hat{A}_1$ and the other is in $\hat{A}_2$. Observe that set $\Delta = (\hat{B}_{\alpha_1\alpha_2}|A_2^*) \cup (\hat{B}_{\alpha_1\alpha_2}|A_3^*) \subseteq \Delta_1$ and sets $(\hat{B}_{\alpha_1\alpha_2}|A_2^*)$ and $(\hat{B}_{\alpha_1\alpha_2}|A_3^*)$ are disjoint (see Figure 3.4). Set $\Delta_2$ can be written as follows $\Delta_2 = (B^*_{\alpha_1\alpha_3} \cup B^*_{\alpha_1\alpha_2}|\hat{A}_1)$ (see Figure 3.4).

After performing the relabel operation $R\langle {A_{1s}}^c, \alpha_2, \hat{f}\rangle$, by the local optimality condition we get a solution of cost no less than the cost of the local optimal solution, or in other words,

$$0 \leq C(\Delta_2) - C(\Delta_1) \leq C(\Delta_2) - C(\Delta) \leq C(B^*_{\alpha_1\alpha_3} \cup B^*_{\alpha_1\alpha_2}|\hat{A}_1) - C(\hat{B}_{\alpha_1\alpha_2}|A_2^*) - C(\hat{B}_{\alpha_1\alpha_2}|A_3^*).$$

This inequality can be rewritten as follows.

$$C(\hat{B}_{\alpha_1\alpha_2}|A_2^*) + C(\hat{B}_{\alpha_1\alpha_2}|A_3^*) \leq C(B^*_{\alpha_1\alpha_3} \cup B^*_{\alpha_1\alpha_2}|\hat{A}_1). \tag{3.5}$$

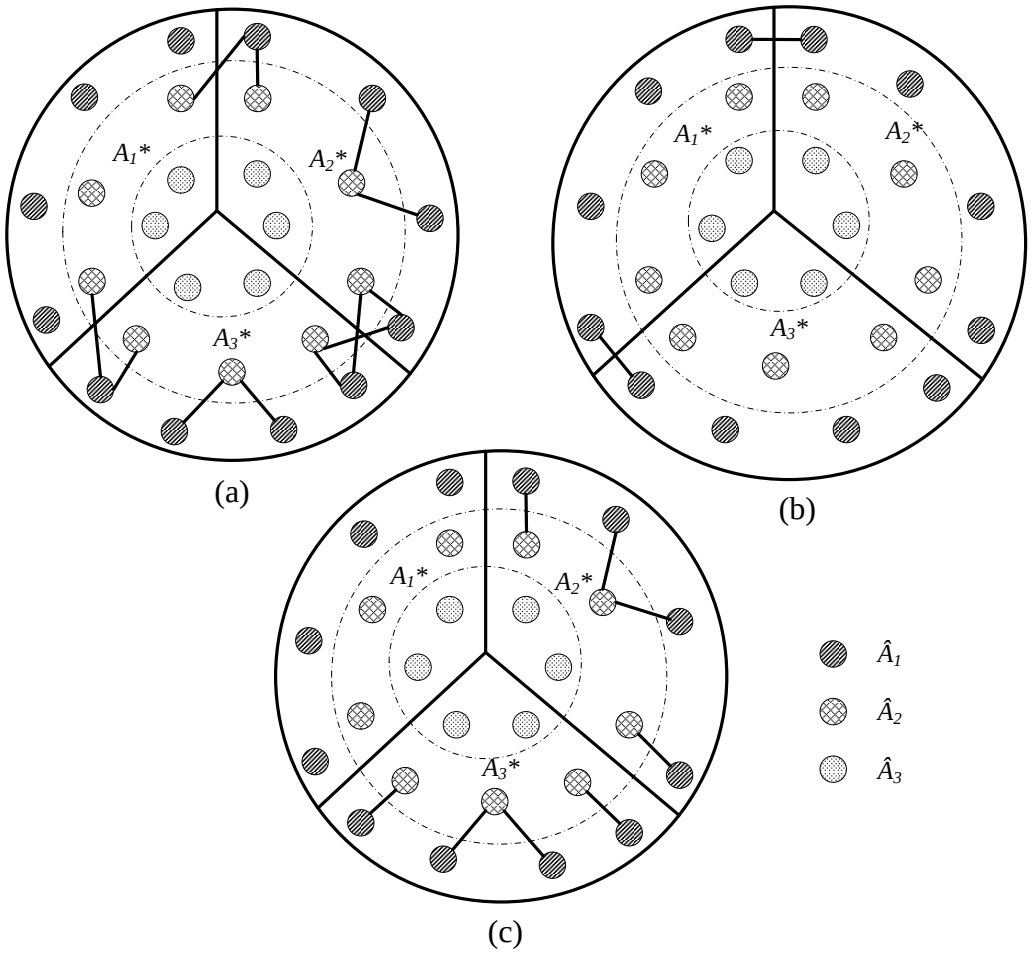


Figure 3.4: In all the figures, partitions $(\hat{A}_1, \hat{A}_2, \hat{A}_3)$ and $(A_1^*, A_2^*, A_3^*)$ are as in Figure 3.3. Figure (a) represents the set $\Delta_1$, Figure (b) represents set $\Delta_2$, and Figure (c) represents set $\Delta$.

We next perform the relabel operation $R\langle {A_1}_s{}^c, \alpha_3, \hat{f}\rangle$. Using similar arguments as above we get the following inequality:

$$C(\hat{B}_{\alpha_1\alpha_3}|A_2^*) + C(\hat{B}_{\alpha_1\alpha_3}|A_3^*) \leq C(B^*_{\alpha_1\alpha_2} \cup B^*_{\alpha_1\alpha_3}|\hat{A}_1). \tag{3.6}$$

To find a bound for the cost of the local optimal solution we need to consider all those edges whose endpoints have different labels in the local optimal solution; therefore, we need to perform more relabel operations: $R\langle {A_2}_s{}^c, \alpha_1, \hat{f}\rangle$, $R\langle {A_2}_s{}^c, \alpha_3, \hat{f}\rangle$, $R\langle {A_3}_s{}^c, \alpha_1, \hat{f}\rangle$ and $R\langle {A_3}_s{}^c, \alpha_2, \hat{f}\rangle$. Using similar arguments as those used to derive (3.5) we get the following inequalities:

$$C(\hat{B}_{\alpha_1\alpha_2}|A_1^*) + C(\hat{B}_{\alpha_1\alpha_2}|A_3^*) \leq C(B^*_{\alpha_1\alpha_2} \cup B^*_{\alpha_2\alpha_3}|\hat{A}_2) \tag{3.7}$$

$$C(\hat{B}_{\alpha_2\alpha_3}|A_1^*) + C(\hat{B}_{\alpha_2\alpha_3}|A_3^*) \leq C(B^*_{\alpha_1\alpha_2} \cup B^*_{\alpha_2\alpha_3}|\hat{A}_2) \tag{3.8}$$

$$C(\hat{B}_{\alpha_1\alpha_3}|A_1^*) + C(\hat{B}_{\alpha_1\alpha_3}|A_2^*) \leq C(B^*_{\alpha_1\alpha_3} \cup B^*_{\alpha_2\alpha_3}|\hat{A}_3) \tag{3.9}$$

$$C(\hat{B}_{\alpha_2\alpha_3}|A_1^*) + C(\hat{B}_{\alpha_2\alpha_3}|A_2^*) \leq C(B^*_{\alpha_1\alpha_3} \cup B^*_{\alpha_2\alpha_3}|\hat{A}_3). \tag{3.10}$$

The sets in the left hand sides of inequalities (3.5)-(3.10) include all the edges in $\hat{S}_p = \hat{S} - S^*$. Therefore, by adding these inequalities we will get a bound for the cost of $\hat{S}_p$. Adding the first term in the left hand sides of inequalities (3.7)-(3.10) we get,

$$\begin{aligned} &C(\hat{B}_{\alpha_1\alpha_2}|A_1^*) + C(\hat{B}_{\alpha_2\alpha_3}|A_1^*) + C(\hat{B}_{\alpha_1\alpha_3}|A_1^*) + C(\hat{B}_{\alpha_2\alpha_3}|A_1^*) = \\ &C(\hat{B}_{\alpha_1\alpha_2} \cup \hat{B}_{\alpha_1\alpha_3} \cup \hat{B}_{\alpha_2\alpha_3}|A_1^*) + C(\hat{B}_{\alpha_2\alpha_3}|A_1^*) \geq C(\hat{B}_{\alpha_1\alpha_2} \cup \hat{B}_{\alpha_1\alpha_3} \cup \hat{B}_{\alpha_2\alpha_3}|A_1^*) = C(\hat{S}|A_1^*). \end{aligned} \tag{3.11}$$

Similarly, adding the second term in the left hand sides of inequalities (3.9) and (3.10), and the first term in the left hand sides of (3.5) and (3.6) we get

$$C(\hat{B}_{\alpha_1\alpha_3}|A_2^*) + C(\hat{B}_{\alpha_2\alpha_3}|A_2^*) + C(\hat{B}_{\alpha_1\alpha_2}|A_2^*) + C(\hat{B}_{\alpha_1\alpha_3}|A_2^*) \geq C(\hat{S}|A_2^*). \tag{3.12}$$

Adding the second term in the left hand sides of (3.5)-(3.8) we get

$$C(\hat{B}_{\alpha_1\alpha_2}|A_3^*) + C(\hat{B}_{\alpha_1\alpha_3}|A_3^*) + C(\hat{B}_{\alpha_1\alpha_2}|A_3^*) + C(\hat{B}_{\alpha_2\alpha_3}|A_3^*) \geq C(\hat{S}|A_3^*). \tag{3.13}$$

In addition, the right hand sides of (3.5) and (3.6) can be replaced with $C(S^*|\hat{A}_1) - C(B^*_{\alpha_2\alpha_3}|\hat{A}_1)$, since

$$\begin{aligned} C(B^*_{\alpha_1\alpha_3} \cup B^*_{\alpha_1\alpha_2}|\hat{A}_1) = C(B^*_{\alpha_1\alpha_2} \cup B^*_{\alpha_1\alpha_3} \cup B^*_{\alpha_2\alpha_3}|\hat{A}_1) - C(B^*_{\alpha_2\alpha_3}|\hat{A}_1) = \\ C(S^*|\hat{A}_1) - C(B^*_{\alpha_2\alpha_3}|\hat{A}_1). \end{aligned} \tag{3.14}$$

Similarly we can replace the right hand sides of (3.7)-(3.10) with the following equivalent terms,

$$C(B^*_{\alpha_1\alpha_2} \cup B^*_{\alpha_2\alpha_3}|\hat{A}_2) = C(S^*|\hat{A}_2) - C(B^*_{\alpha_1\alpha_3}|\hat{A}_2) \tag{3.15}$$

$$C(B^*_{\alpha_1\alpha_3} \cup B^*_{\alpha_2\alpha_3}|\hat{A}_3) = C(S^*|\hat{A}_3) - C(B^*_{\alpha_1\alpha_2}|\hat{A}_3). \tag{3.16}$$

Adding (3.5)-(3.10) and using (3.11)-(3.16) to simplify we get

$$\begin{aligned} &C(\hat{S}|A_1^*) + C(\hat{S}|A_2^*) + C(\hat{S}|A_3^*) \leq \\ &2\left[C(S^*|\hat{A}_1) + C(S^*|\hat{A}_2) + C(S^*|\hat{A}_3) - \left(C(B^*_{\alpha_2\alpha_3}|\hat{A}_1) + C(B^*_{\alpha_1\alpha_3}|\hat{A}_2) + C(B^*_{\alpha_1\alpha_2}|\hat{A}_3)\right)\right]. \end{aligned} \tag{3.17}$$

This last inequality can be re-written as:

$$C(\hat{S}_p) \leq 2\left[C(S_p^*) - \left(C(B^*_{\alpha_2\alpha_3}|\hat{A}_1) + C(B^*_{\alpha_1\alpha_3}|\hat{A}_2) + C(B^*_{\alpha_1\alpha_2}|\hat{A}_3)\right)\right]. \tag{3.18}$$

Adding $C(\hat{S} \cap S^*)$ to the left hand side and $2C(\hat{S} \cap S^*)$ to the right hand side of this last inequality we get our first bound for the cost of the local optimal solution,

$$C(\hat{S}) \leq 2\left[C(S^*) - \left(C(B^*_{\alpha_2\alpha_3}|\hat{A}_1) + C(B^*_{\alpha_1\alpha_3}|\hat{A}_2) + C(B^*_{\alpha_1\alpha_2}|\hat{A}_3)\right)\right]. \tag{3.19}$$

### 3.4.2 Second Bound

In order to obtain a second bound for the cost of the local optimal solution we proceed similarly as before but select different sets of nodes on which to perform relabel operations. This time we choose to change the labels of the nodes in the partitions $A_1^*$, $A_2^*$ and $A_3^*$. Note that we still change labels assigned by the local optimum solution to the selected nodes.

We now show that if we perform the relabel operations $R\langle A_i^*, \alpha_i, \hat{f}\rangle$ for $1 \leq i \leq 3$ we can find an upper bound for the cost of all the edges in $(\hat{S}|A_1^*)$, $(\hat{S}|A_2^*)$ and $(\hat{S}|A_3^*)$. Adding these three costs we get a bound for $C(\hat{S}_p)$.

First, we perform the relabel operation $R\langle A_1^*, \alpha_1, \hat{f}\rangle$. This operation decreases the contribution to the cost of the solution made by $\Theta_1 = (\hat{B}_{\alpha_1\alpha_2} \cup \hat{B}_{\alpha_1\alpha_3} \cup \hat{B}_{\alpha_2\alpha_3}|A_1^*) = (\hat{S}|A_1^*)$. It also decreases the contribution to the cost of the solution made by the edges in set $\Theta_2 = (\hat{S} \cap S^*|A_1^* : \hat{A}_1 \cap (A_2^* \cup A_3^*))$. Let $\Theta_3 = (B^*_{\alpha_1\alpha_2} \cup B^*_{\alpha_1\alpha_3}|\hat{A}_2)$ and $\Theta_4 = (B^*_{\alpha_1\alpha_2} \cup B^*_{\alpha_1\alpha_3}|\hat{A}_3)$. After applying the relabel operation the cost of the solution is increased by the cost of edges in $\Theta_3$ and $\Theta_4$. Note that sets $\Theta_1$, $\Theta_2$, $\Theta_3$ and $\Theta_4$ are disjoint. After performing the relabel operation $R\langle A_1^*, \alpha_1, \hat{f}\rangle$, by the local optimality property we get

$$C(\Theta_3) + C(\Theta_4) - C(\Theta_1) - C(\Theta_2) =$$
$$C(B^*_{\alpha_1\alpha_2} \cup B^*_{\alpha_1\alpha_3}|\hat{A}_2) + C(B^*_{\alpha_1\alpha_2} \cup B^*_{\alpha_1\alpha_3}|\hat{A}_3) - C(\hat{S}|A_1^*) - C(\hat{S} \cap S^*|A_1^* : \hat{A}_1 \cap (A_2^* \cup A_3^*)) \geq 0.$$

This inequality can be rewritten as follows since $C(\hat{S} \cap S^*|A_1^* : \hat{A}_1 \cap (A_2^* \cup A_3^*)) \geq 0$:

$$C(\hat{S}|A_1^*) \leq C(B^*_{\alpha_1\alpha_2} \cup B^*_{\alpha_1\alpha_3}|\hat{A}_2) + C(B^*_{\alpha_1\alpha_2} \cup B^*_{\alpha_1\alpha_3}|\hat{A}_3). \tag{3.20}$$

We perform two more relabel operations: $R\langle A_2^*, \alpha_2, \hat{f}\rangle$ and $R\langle A_3^*, \alpha_3, \hat{f}\rangle$. Using a similar argument as in (3.20) we get,

$$C(\hat{S}|A_2^*) \leq C(B^*_{\alpha_1\alpha_2} \cup B^*_{\alpha_2\alpha_3}|\hat{A}_1) + C(B^*_{\alpha_1\alpha_2} \cup B^*_{\alpha_2\alpha_3}|\hat{A}_3) \tag{3.21}$$

$$C(\hat{S}|A_3^*) \leq C(B^*_{\alpha_1\alpha_3} \cup B^*_{\alpha_2\alpha_3}|\hat{A}_1) + C(B^*_{\alpha_1\alpha_3} \cup B^*_{\alpha_2\alpha_3}|\hat{A}_2). \tag{3.22}$$

Adding the left sides of (3.20)-(3.22) we get,

$$C(\hat{S}|A_1^*) + C(\hat{S}|A_2^*) + C(\hat{S}|A_3^*) = C(\hat{S}_p). \tag{3.23}$$

Adding the first term in the right hand side of (3.21) and (3.22) we get,

$$C(B^*_{\alpha_1\alpha_2} \cup B^*_{\alpha_2\alpha_3}|\hat{A}_1) + C(B^*_{\alpha_1\alpha_3} \cup B^*_{\alpha_2\alpha_3}|\hat{A}_1) = C(B^*_{\alpha_1\alpha_2} \cup B^*_{\alpha_1\alpha_3} \cup B^*_{\alpha_2\alpha_3}|\hat{A}_1) + C(B^*_{\alpha_2\alpha_3}|\hat{A}_1) =$$
$$C(S^*|\hat{A}_1) + C(B^*_{\alpha_2\alpha_3}|\hat{A}_1) \tag{3.24}$$

Similarly,

$$C(B^*_{\alpha_1\alpha_2} \cup B^*_{\alpha_1\alpha_3}|\hat{A}_2) + C(B^*_{\alpha_1\alpha_3} \cup B^*_{\alpha_2\alpha_3}|\hat{A}_2) = C(S^*|\hat{A}_2) + C(B^*_{\alpha_1\alpha_3}|\hat{A}_2) \tag{3.25}$$

and,

$$C(B^*_{\alpha_1\alpha_2} \cup B^*_{\alpha_1\alpha_3}|\hat{A}_3) + C(B^*_{\alpha_1\alpha_2} \cup B^*_{\alpha_2\alpha_3}|\hat{A}_3) = C(S^*|\hat{A}_3) + C(B^*_{\alpha_1\alpha_2}|\hat{A}_3). \tag{3.26}$$

Adding the first term in the right sides of (3.24)-(3.26) we get,

$$C(S^*|\hat{A}_1) + C(S^*|\hat{A}_2) + C(S^*|\hat{A}_3) = C(S_p^*). \tag{3.27}$$

Adding (3.20)-(3.22) and then using (3.23)-(3.27) to simplify we get,

$$C(\hat{S}_p) \leq C(S_p^*) + \left[C(B^*_{\alpha_2\alpha_3}|\hat{A}_1) + C(B^*_{\alpha_1\alpha_3}|\hat{A}_2) + C(B^*_{\alpha_1\alpha_2}|\hat{A}_3)\right]. \tag{3.28}$$

Finally, adding $C(\hat{S} \cap S^*)$ to both sides of (3.28) we get our second bound,

$$C(\hat{S}) \leq C(S^*) + \left[C(B^*_{\alpha_2\alpha_3}|\hat{A}_1) + C(B^*_{\alpha_1\alpha_3}|\hat{A}_2) + C(B^*_{\alpha_1\alpha_2}|\hat{A}_3)\right]. \tag{3.29}$$

### 3.4.3 Computing the Approximation Ratio

We use inequalities (3.19) and (3.29) to determine the approximation ratio of our algorithm for the 3-way cut problem. We consider two cases:

i. If $C(B^*_{\alpha_2\alpha_3}|\hat{A}_1) + C(B^*_{\alpha_1\alpha_3}|\hat{A}_2) + C(B^*_{\alpha_1\alpha_2}|\hat{A}_3) > \frac{1}{3}C(S^*)$, then

$$C(S^*) - \left[C(B^*_{\alpha_2\alpha_3}|\hat{A}_1) + C(B^*_{\alpha_1\alpha_3}|\hat{A}_2) + C(B^*_{\alpha_1\alpha_2}|\hat{A}_3)\right] \leq \frac{2}{3}C(S^*).$$

Therefore (3.19) simplifies to:

$$C(\hat{S}) \leq \frac{4}{3}C(S^*). \tag{3.30}$$

ii. If $C(B^*_{\alpha_2\alpha_3}|\hat{A}_1) + C(B^*_{\alpha_1\alpha_3}|\hat{A}_2) + C(B^*_{\alpha_1\alpha_2}|\hat{A}_3) \leq \frac{1}{3}C(S^*)$, then Inequality (3.29) simplifies to:

$$C(\hat{S}) \leq \frac{4}{3}C(S^*). \tag{3.31}$$

Therefore, by (3.30) and (3.31)

$$C(\hat{S}) \leq \frac{4}{3}C(S^*). \tag{3.32}$$

**THEOREM 6** *The approximation ratio of algorithm MULTIWAY CUT for the 3-way cut problem is* $\frac{4}{3}$.

## 3.5 The Multiway Cut Problem

We extend the definitions of $\hat{A}_i$, $A_i^*$, $\hat{B}_{\alpha_i\alpha_j}$ and $B^*_{\alpha_i\alpha_j}$ for $k$ labels in the natural way. Similarly as in Section 3.4 we perform several relabel operations and then use the local optimality property to obtain a bound for $C(\hat{S})$.

### 3.5.1 First Bound

To bound the cost of the solution produced by our algorithm we perform several relabel operations on similar sets of nodes as those used in Section 3.4.1. Let $A_{is} = A_i^* \cap \hat{A}_i$ and ${A_{is}}^c = \hat{A}_i - A_{is}$ for $i \in I = \{1, 2, 3, ..., k\}$.

First we perform the relabel operation $R\langle {A_{1s}}^c, \alpha_2, \hat{f}\rangle$. This operation decreases the cost of the solution by the cost of edges in set $\Gamma_1 = (\hat{B}_{\alpha_1\alpha_2}|\hat{A}_1 \cap (\bigcup_{i\in I-\{1\}} A_i^*) : \hat{A}_2)$. In addition, the cost of the solution increases by the cost of the edges in set $\Gamma_2 = \bigcup_{i\in I-\{1\}}(B^*_{\alpha_1\alpha_i}|\hat{A}_1)$. Note that $\Gamma = \bigcup_{i\in I-\{1\}}(\hat{B}_{\alpha_1\alpha_2}|A_i^*) = \bigcup_{i\in I-\{1\}}(\hat{B}_{\alpha_1\alpha_2}|\hat{A}_1 \cap A_i^* : \hat{A}_2 \cap A_i^*) \subseteq \Gamma_1$.

After performing $R\langle {A_{1s}}^c, \alpha_2, \hat{f}\rangle$, by the local optimality property the cost of the new solution is greater than or equal to the cost of the local optimal solution; therefore,

$$0 \leq C(\Gamma_2) - C(\Gamma_1) \leq C(\Gamma_2) - C(\Gamma) = \sum_{i\in I-\{1\}} C(B^*_{\alpha_1\alpha_i}|\hat{A}_1) - \sum_{i\in I-\{1\}} C(\hat{B}_{\alpha_1\alpha_2}|A_i^*).$$

The last equality holds because the sets $(B^*_{\alpha_1\alpha_i}|\hat{A}_1)$, $i \in I-1$ are disjoint and so are the sets $(\hat{B}_{\alpha_1\alpha_2}|A_i^*)$, $i \in I-1$. This inequality can be rewritten as follows,

$$\sum_{i\in I-\{1\}} C(\hat{B}_{\alpha_1\alpha_2}|A_i^*) \leq \sum_{i\in I-\{1\}} C(B^*_{\alpha_1\alpha_i}|\hat{A}_1). \tag{3.33}$$

We perform further relabel operations $R\langle {A_{1s}}^c, \alpha_l, \hat{f}\rangle$ for all $l \in I - \{1, 2\}$. Using similar arguments as above we get the following inequalities,

$$\sum_{i\in I-\{1\}} C(\hat{B}_{\alpha_1\alpha_l}|A_i^*) \leq \sum_{i\in I-\{1\}} C(B^*_{\alpha_1\alpha_i}|\hat{A}_1) \tag{3.34}$$

for all $3 \leq l \leq k$.

To find a bound for the cost of the remaining edges in the local optimal solution we perform the following additional relabel operations: $R\langle {A_{hs}}^c, j, \hat{f}\rangle$ for all $h \in I - \{1\}$, $j \in I - \{h\}$. Using similar arguments as in (3.33) we get the following inequalities,

$$\sum_{i\in I-\{h\}} C(\hat{B}_{\alpha_h\alpha_j}|A_i^*) \leq \sum_{i\in I-\{h\}} C(B^*_{\alpha_h\alpha_i}|\hat{A}_h) \tag{3.35}$$

for all $h \in I-\{1\}$ and $j \in I-\{h\}$. Adding inequalities (3.33) and (3.34) for all $3 \leq l \leq k$, and (3.35) for all $h \in I - \{1\}$ and $j \in I - \{h\}$ we get,

$$\sum_{h\in I}\sum_{j\in I-\{h\}}\sum_{i\in I-\{h\}} C(\hat{B}_{\alpha_h\alpha_j}|A_i^*) \leq (k-1)\sum_{h\in I}\sum_{i\in I-\{h\}} C(B^*_{\alpha_h\alpha_i}|\hat{A}_h).$$

The above inequality can be rewritten as follows,

$$\sum_{i\in I}\sum_{h\in I-\{i\}}\sum_{j\in I-\{h\}} C(\hat{B}_{\alpha_h\alpha_j}|A_i^*) \leq (k-1)\sum_{h\in I}\sum_{i\in I-\{h\}} C(B^*_{\alpha_h\alpha_i}|\hat{A}_h). \tag{3.36}$$

To simplify (3.36), we first consider the left side and note that for each value $i \in I$,

$$\sum_{h \in I-\{i\}} \sum_{j \in I-\{h\}} C(\hat{B}_{\alpha_h \alpha_j} | A_i^*) \geq \sum_{1 \leq r < s \leq k} C(\hat{B}_{\alpha_r \alpha_s} | A_i^*). \tag{3.37}$$

because $\hat{B}_{\alpha_r \alpha_s} = \hat{B}_{\alpha_s \alpha_r}$ for all $1 \leq r < s \leq k$, and even though the left hand side is missing the terms $C(\hat{B}_{\alpha_i \alpha_j} | A_i^*)$, $j > i$ which are included in the sum in the right hand side, it includes the terms $C(\hat{B}_{\alpha_j \alpha_i} | A_i^*)$, $j > i$.
Also observe that

$$\sum_{1 \leq r < s \leq k} C(\hat{B}_{\alpha_r \alpha_s} | A_i^*) = C(\bigcup_{1 \leq r < s \leq k} \hat{B}_{\alpha_r \alpha_s} | A_i^*) = C(\hat{S} | A_i^*) \tag{3.38}$$

because the sets $(\hat{B}_{\alpha_r \alpha_s} | A_i^*)$, $1 \leq r < s \leq k$, $i \in I$ are disjoint.

To simplify the right side of (3.36) we consider the following equality,

$$\sum_{i \in I-\{h\}} C(B^*_{\alpha_h \alpha_i} | \hat{A}_h) = C(\bigcup_{i \in I-\{h\}} B^*_{\alpha_h \alpha_i} | \hat{A}_h) = C(\bigcup_{1 \leq r < s \leq k} B^*_{\alpha_r \alpha_s} | \hat{A}_h) - C(\bigcup_{\substack{1 \leq r < s \leq k \\ r,s \neq h}} B^*_{\alpha_r \alpha_s} | \hat{A}_h) =$$
$$C(S^* | \hat{A}_h) - \sum_{\substack{1 \leq r < s \leq k \\ r,s \neq h}} C(B^*_{\alpha_r \alpha_s} | \hat{A}_h). \tag{3.39}$$

Simplifying (3.36) using (3.37)-(3.39) we get,

$$\sum_{i \in I} C(\hat{S} | A_i^*) \leq (k-1) \sum_{h \in I} \left[ C(S^* | \hat{A}_h) - \sum_{\substack{1 \leq r < s \leq k \\ r,s \neq h}} C(B^*_{\alpha_r \alpha_s} | \hat{A}_h) \right] \tag{3.40}$$

that can be re-written as:

$$C(\hat{S}_p) \leq (k-1) \left[ C(S_p^*) - \sum_{h \in I} \sum_{\substack{1 \leq r < s \leq k \\ r,s \neq h}} C(B^*_{\alpha_r \alpha_s} | \hat{A}_h) \right]. \tag{3.41}$$

Adding $C(\hat{S} \cap S^*)$ to the left side and $(k-1)C(\hat{S} \cap S^*)$ to the right side of the above inequality we get our first bound for $C(\hat{S})$,

$$C(\hat{S}) \leq (k-1) \left[ C(S^*) - \sum_{h \in I} \sum_{\substack{1 \leq r < s \leq k \\ r,s \neq h}} C(B^*_{\alpha_r \alpha_s} | \hat{A}_h) \right]. \tag{3.42}$$

### 3.5.2 Second bound

Proceeding as in Section 3.4 to find a second bound for the cost of the local optimal solution we perform relabel operations $R\langle A_i^*, \alpha_i, \hat{f}\rangle$ for all $i \in I = \{1, 2, ..., k\}$ and then proceed similarly as in Section 3.4 by adding all the cost-change inequalities related to these operations to get another bound for $C(\hat{S}_p)$.

First let us perform the relabel operation $R\langle A_1^*, \alpha_1, \hat{f}\rangle$. This operation decreases the cost of the solution by the cost of the edges in set $\Lambda_1 = (\hat{S}|A_1^*)$. It also decreases the cost of the solution by the cost of the edges in set $\Lambda_2 = (\hat{S} \cap S^*|A_1^* : \hat{A}_1 \cap (\bigcup_{j\in I-\{1\}} A_j^*))$. In addition the cost of the solution increases by the cost of the edges in set $\Lambda_3 = \bigcup_{h\in I-\{1\}}(\bigcup_{j\in I-\{1\}} B^*_{\alpha_1\alpha_j}|\hat{A}_h)$ as before performing the above relabel operation both endpoints of the edges in $(\bigcup_{j\in I-\{1\}} B^*_{\alpha_1\alpha_j}|\hat{A}_h)$ were labelled $\alpha_h$ and after the operation one endpoint of each of these edges is labelled $\alpha_1$ for all $h \in I - \{1\}$. Hence, by the local optimality property we get,

$$C(\Lambda_3) - C(\Lambda_1) - C(\Lambda_2) = \\ C(\bigcup_{h\in I-\{1\}} (\bigcup_{j\in I-\{1\}} B^*_{\alpha_1\alpha_j}|\hat{A}_h)) - C(\hat{S}|A_1^*) - C(\hat{S} \cap S^*|A_1^* : \hat{A}_1 \cap (\bigcup_{j\in I-\{1\}} A_j^*)) \geq 0.$$

This inequality can be rewritten as follows, since $C(S' \cap S^*|A_1^* : \hat{A}_1 \cap (\bigcup_{i\in I-\{1\}} A_i^*)) \geq 0$ and sets $(\bigcup_{j\in I-\{1\}} B^*_{\alpha_1\alpha_j}|\hat{A}_h)$ are disjoint for all $h \in I - \{1\}$:

$$C(\hat{S}|A_1^*) \leq \sum_{h\in I-\{1\}} C(\bigcup_{j\in I-\{1\}} B^*_{\alpha_1\alpha_j}|\hat{A}_h).$$

We now perform the relabel operations $R\langle A_i^*, \alpha_i, \hat{f}\rangle$ for all $i \in I$. Using similar arguments as above, we get the following inequality for the cost changes related to the relabel operation $R\langle A_i^*, \alpha_i, \hat{f}\rangle$,

$$C(\hat{S}|A_i^*) \leq \sum_{h\in I-\{i\}} C(\bigcup_{j\in I-\{i\}} B^*_{\alpha_i\alpha_j}|\hat{A}_h). \tag{3.43}$$

Adding inequality (3.43) for all $i \in I$ we get,

$$\sum_{i\in I} C(\hat{S}|A_i^*) \leq \sum_{i\in I} \sum_{h\in I-\{i\}} C(\bigcup_{j\in I-\{i\}} B^*_{\alpha_i\alpha_j}|\hat{A}_h).$$

that can be rewritten as follows,

$$\sum_{i\in I} C(\hat{S}|A_i^*) \leq \sum_{h\in I} \sum_{i\in I-\{h\}} C(\bigcup_{j\in I-\{i\}} B^*_{\alpha_i\alpha_j}|\hat{A}_h). \tag{3.44}$$

Recall that,

$$\sum_{i\in I} C(\hat{S}|A_i^*) = C(\hat{S}_p). \tag{3.45}$$

Since sets $(B^*_{\alpha_i\alpha_j}|\hat{A}_h)$ are disjoint for all $j \in I - \{i\}$, $h \in I$ and $\hat{B}_{\alpha_r\alpha_s} = \hat{B}_{\alpha_s\alpha_r}$ for all $1 \leq r < s \leq k$ then,

$$\sum_{i\in I-\{h\}} C(\bigcup_{j\in I-\{i\}} B^*_{\alpha_i\alpha_j}|\hat{A}_h) = \sum_{i\in I-\{h\}}\sum_{j\in I-\{i\}} C(B^*_{\alpha_i\alpha_j}|\hat{A}_h) =$$

$$\sum_{i\in I}\sum_{j\in I-\{i\}} C(B^*_{\alpha_i\alpha_j}|\hat{A}_h) - \sum_{j\in I-\{h\}} C(B^*_{\alpha_h\alpha_j}|\hat{A}_h) = 2\sum_{1\leq r<s\leq k} C(B^*_{\alpha_r\alpha_s}|\hat{A}_h) - \sum_{j\in I-\{h\}} C(B^*_{\alpha_h\alpha_j}|\hat{A}_h) =$$

$$\sum_{1\leq r<s\leq k} C(B^*_{\alpha_r\alpha_s}|\hat{A}_h) + \sum_{\substack{1\leq r<s\leq k\\ r,s\neq h}} C(B^*_{\alpha_r\alpha_s}|\hat{A}_h) \quad (3.46)$$

and furthermore,

$$\sum_{1\leq r<s\leq k} C(B^*_{\alpha_r\alpha_s}|\hat{A}_h) = C(\bigcup_{1\leq r<s\leq k} B^*_{\alpha_r\alpha_s}|\hat{A}_h) = C(S^*|\hat{A}_h). \quad (3.47)$$

Simplifying (3.44) using (3.45)-(3.47) we get,

$$C(\hat{S}_p) \leq \sum_{h\in I}\left[C(S^*|\hat{A}_h) + \sum_{\substack{1\leq r<s\leq k\\ r,s\neq h}} C(B^*_{\alpha_r\alpha_s}|\hat{A}_h)\right] = C(S^*_p) + \sum_{h\in I}\sum_{\substack{1\leq r<s\leq k\\ r,s\neq h}} C(B^*_{\alpha_r\alpha_s}|\hat{A}_h).$$

Adding $C(\hat{S}\cap S^*)$ to both sides of the above inequality we get our second bound,

$$C(\hat{S}) \leq C(S^*) + \sum_{h\in I}\sum_{\substack{1\leq r<s\leq k\\ r,s\neq h}} C(B^*_{\alpha_r\alpha_s}|\hat{A}_h). \quad (3.48)$$

### 3.5.3 Computing the Approximation Ratio

Similarly as in Section 3.4.3 we consider two cases,

i. If $\sum_{h\in I}\sum_{\substack{1\leq r<s\leq k\\ r,s\neq h}} C(B^*_{\alpha_r\alpha_s}|\hat{A}_h) > \frac{k-2}{k}C(S^*)$, then

$$C(S^*) - \sum_{h\in I}\sum_{\substack{1\leq r<s\leq k\\ r,s\neq h}} C(B^*_{\alpha_r\alpha_s}|\hat{A}_h) < \frac{2}{k}C(S^*). \quad (3.49)$$

Therefore by (3.42),

$$C(\hat{S}) < \frac{2k-2}{k}C(S^*). \quad (3.50)$$

ii. If $\sum_{h\in I}\sum_{\substack{1\leq r<s\leq k\\ r,s\neq h}} C(B^*_{\alpha_r\alpha_s}|\hat{A}_h) \leq \frac{k-2}{k}C(S^*)$, then (3.48) simplifies to:

$$C(\hat{S}) \leq \frac{2k-2}{k}C(S^*). \quad (3.51)$$

Thus, by (3.50) and (3.51)

$$C(\hat{S}) \leq \left[2 - \frac{2}{k}\right] C(S^*). \tag{3.52}$$

**THEOREM 7** *The approximation ratio of algorithm MULTIWAY CUT is* $2 - \frac{2}{k}$.

## 3.6 Tight Example

Figure 3.5 shows an example proving the tightness of the analysis. The local optimal solution assigns label $\alpha_i$ to terminal $x_i$ for $i = 1, 2, ..., k-1$ and it assigns label $\alpha_k$ to all the other nodes. The global optimal solution assigns label $\alpha_i$ to each pair of nodes $x_i$, $y_i$, for $i = 1, 2, ..., k$.

The cost of the local optimal solution is $2w(k-1)$ and the cost of the global optimal solution is $wk$, therefore the approximation ratio is $2-\frac{2}{k}$. To show that the first solution in Figure 3.5 is a local optimal solution we consider all possible relabel operations $R\langle Y, \alpha_i, \hat{f}\rangle$ for each $Y \subseteq \{y_1, y_2, ..., y_k\}$ and $i = 1, 2, ..., k$. If $y_i \notin Y$ then the cost of the solution will increase as some of the edges of the form $(y_i, y_{i+1})$, $(x_k, y_1)$ or $(x_k, y_k)$ will now contribute to the cost of the solution. If $y_i \in Y$ then the cost of the edge $(x_i, y_i)$ will not be part of the cost of the solution, but at least two edges of cost $w$ will now contribute to the cost of the solution and, thus, the cost of the solution cannot decrease.

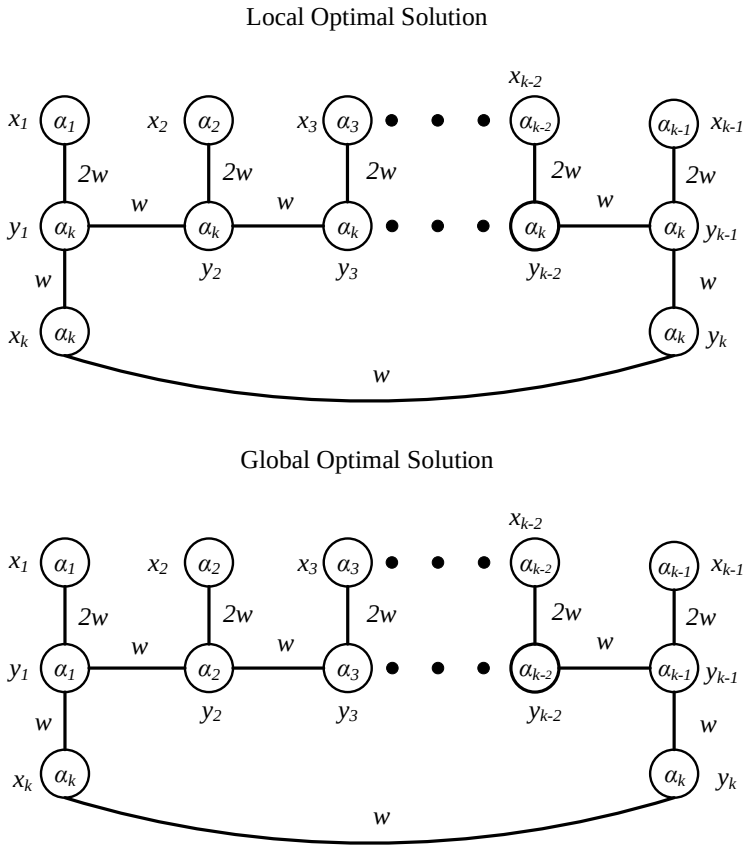


Figure 3.5: Node labels are inside the nodes and node names are written beside the nodes. Edge costs are written beside the edges. The terminals are $x_1, x_2, \ldots, x_k$.

## 3.7 Variations of the Multiway Cut Problem

By making minor changes to the algorithm used for finding a minimum cost labeling we can use our local search algorithm on two variations of the multiway cut problem: (i) when certain sets of nodes need to be in the same partition, and (ii) when some nodes must belong to only certain partitions.

These versions of the the problem arise in computer vision when we want to segment an image and we either know that certain pixels have to be in the same segment or we know that some pixels can only belong to certain segments.

### 3.7.1 Nodes that Need to be in the Same Partition

In this variation of the multiway cut problem we are given a graph $G = (V, E)$, a set of terminals $T = \{t_1, t_2, ..., t_k\} \subseteq V$ and a collection $F = \{q_1, q_2, ...q_h\}$ of subsets of nodes. Sets $q_i \subseteq V$, $i = 1, 2, ..., h$, are disjoint and $q_i$ does not include any terminals. The goal is to find a minimum weight set $E' \subseteq E$ such that removing $E'$ from $G$ divides it into $k$ partitions so that each partition includes exactly one terminal and all nodes in set $q_i$ are in the same partition, for all $i = 1, 2, ..., h$.

First we transform graph $G = (V, E)$ into a new graph $G' = (V', E')$ as described below. To find a minimum cost labeling for this version of the problem we compute a minimum cut in a graph $G'_\alpha = (V_\alpha, E_\alpha)$ built from $G'$ as described in Section 3.3. The transformation of $G$ to $G'$ is as follows,

**Algorithm 5** Transform $G = (V, E)$

**Input:** Graph $G = (V, E)$, collection $F = \{q_1, q_2, ...q_h\}$ of subset of vertices
**Output:** Graph $G' = (V', E')$
$G' \leftarrow G$
**for** $i \leftarrow 1$ **to** $h$ **do**
    $N_{q_i} \leftarrow \bigcup_{v \in q_i} N_v$, where $N_v$ is the set of neighbors of node $v$ in $G'$
    Merge all nodes in $q_i$ into one super-node $Q_i$, so the set of neighbors of $Q_i$ includes all the
    nodes in $N_{q_i} \setminus q_i$.
    For all $u \in N_{q_i} \setminus q_i$ add edge $(u, Q_i)$ and set
    $cost(u, Q_i) = \sum_{v \in N_u \cap q_i} cost(u, v)$.
**end for**
return $G'$

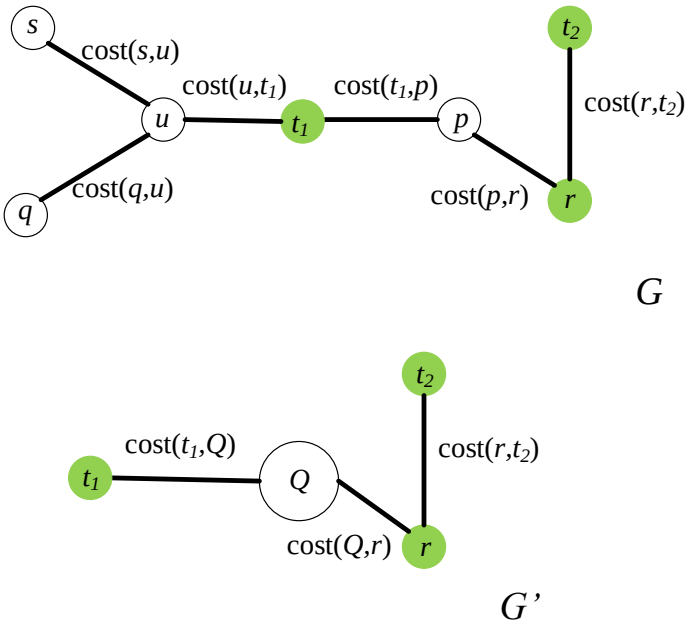


Figure 3.6: An example of the transformation of $G$ into $G'$. Nodes that need to be in the same partition are $s, q, u, p$ so in the transformed graph $G'$ a super node $Q$ is created for these nodes. Also, $cost(t_1, Q) = cost(u, t_1) + cost(t_1, p)$ and $cost(Q, r) = cost(p, r)$.

Note that there is a 1-1 correspondence between the feasible multiway cuts $m$ in $G$ that separate the terminals and keep the vertices of each set $q_i$ in the same partitions,

and the feasible multiway cuts $m'$ in $G'$ that separate the $k$ terminals. Furthermore, a feasible multiway cut $m$ of $G$ and its corresponding feasible multiway cut $m'$ in $G'$ have the same cost because,

- If $\{(u,v) \mid u \notin \bigcup_{1\leq j\leq h} q_j, v \in q_i, i \in \{1,2,...,h\}\} \subseteq m$, then $(u,Q_i) \in m'$ and $cost(u,Q_i) = \sum_{v \in N_u \cap q_i} cost(u,v)$.
- If $\{(u,v) \mid u \in q_i, v \in q_j, i \neq j \in \{1,2,...,h\}\} \subseteq m$, then $(Q_i,Q_j) \in m'$. In addition, observe that algorithm Transform sets $cost(Q_i,Q_j) = \sum_{u\in q_i} \sum_{v \in N_u \cap q_j} cost(u,v)$.

Since there is a 1-1 mapping between the feasible multiway cuts in $G$ and the feasible multiway cuts in $G'$ with corresponding cuts having the same cost and the construction of $G'_\alpha$ from $G'$ is exactly as described in Section 3.3, then the same analysis in Section 3.5 still hold for this version of the multiway cut problem and so we can also guarantee a $2-\frac{2}{k}$ approximation ratio for this problem.

### 3.7.2 Nodes that Can Only be in Some Partitions

In this variation of the multiway cut problem we are given a graph $G=(V,E)$, a set of terminals $T=\{t_1,t_2,...,t_k\} \subseteq V$, a set of labels $L=\{l_1,l_2,...,l_k\}$, and also for each node $v \in V \setminus T$, we are given a set $l_v \subseteq L$ of allowed labels for it. Each terminal needs to be assigned a unique label. The goal is to assign labels to the nodes in such a way that the sum of the weights of the edges for which their endpoints have different labels is minimum and each node is assigned a valid label.

We can modify our local search algorithm so it works on this version of the multiway cut problem as well. The construction of the graph $G_\alpha$ described in Section 3.3 needs to be changed as follows,

- For each node $v$ in $G$, if $\alpha$ is one of the possible labels that node $v$ can have then the weight of edge $(\alpha,v)$ in $G_\alpha$ is 0, otherwise if $v$ cannot be labelled $\alpha$ then we set the weight of edge $(\alpha,v)$ to $\infty$.

### Analysis

The analysis in Section 3.5 does not work for this version of the problem because to compute the bound in Section 3.5.1 we need to assign to some nodes in the local optimal solution all possible labels, but this is might not be allowed in this version of the problem.

However, we can still obtain the bound computed in Section 3.5.2 to get Inequality (3.48). This is because for computing this bound we only change the labels of the nodes in $A_i^*$ to $\alpha_i$ and because these nodes are labeled $\alpha_i$ in the global optimal solution the above is a valid relabeling. Therefore, proceeding as in Section 3.5.2 we get,

$$C(\hat{S}) \leq C(S^*) + \sum_{h\in I} \sum_{\substack{1\leq r<s\leq k \\ r,s\neq h}} C(B^*_{\alpha_r\alpha_s}|\hat{A}_h). \tag{3.53}$$

Since $\sum_{h \in I} \sum_{\substack{1 \leq r < s \leq k \\ r,s \neq h}} C(B^*_{\alpha_r \alpha_s} | \hat{A}_h) \leq C(S^*)$ we can rewrite (3.53) as follows,

$$C(\hat{S}) \leq 2C(S^*) \tag{3.54}$$

Therefore, we can guarantee an approximation ratio 2 for the version of the multiway cut problem when certain nodes can only be in some partitions.

## 3.8 Experimental Results

We compared our local search algorithm with four other approximation algorithms for the multiway cut problem: Dahlhaus et al.'s isolation heuristic [8], the algorithm of Calinescu et al. [6], the algorithm of Sharma and Vondrak [16], and the algorithm of Buchbinder et al. [4, 5].

We implemented MULTIWAY CUT and the other approximation algorithms using Java 1.8. The commercial integer and linear program solver Cplex 12.7, configured using default settings, was used to compute optimal solutions for each test instance. The experiments were performed on a computer using an Intel Core i5-5200U 220GHz (4 CPUs) with 16GB of RAM and SHARCNET's high performance computing clusters: Orca, using an AMD Opteron 2.2GHz (4 CPUs) with 32GB of RAM, Saw, using an Intel Xeon 2.83Ghz (4 CPUs) with 16GB of RAM, and Kraken, using an AMD Opteron 2.2GHz (4 CPUs) with 8GB of RAM.

We used the maximum flow algorithm of Goldberg and Tarjan to compute minimum cost relabel operations in our local search algorithm. To ensure a polynomial running time for our algorithm as discussed at the end of Section 3.2, for most of our experiments we chose the value of $\epsilon$ such that $(1 - \frac{\epsilon}{k^2})$ was equal to 99/100.

The isolation heuristic of Dahlhaus et al.'s [8] shares the same worst case approximation ratio as MULTIWAY CUT of $2 - \frac{2}{k}$, but as we show, its experimental performance is much worse than that of our local search algorithm. Cplex is used to solve the linear programs needed for the algorithms of Calinescu et al. [6], Sharma and Vondrak [16], and Buchbinder et al. [4, 5].

### 3.8.1 Input Data

We used a variety of inputs obtained from network benchmarks used in DIMACS competitions (https://dimacs.rutgers.edu/Challenges/) and randomly generated networks. Even though past DIMACS competitions have not included the multiway cut problem in their implementation challenges, we were able to use network benchmarks for maximum clique, maximum independent set, Hamming instances, Keller's conjecture instances, p-hat generated instances, and Steiner tree instances by considering only the largest connected component in each graph and choosing, when needed, random edge capacities and terminals for each instance.

The structure of a graph impacts how well a multiway cut algorithm performs. When edges incident on the terminals have much smaller capacities than edges not incident on them, an optimal solution will simply select the edges incident on the terminals.

Furthermore, when the number of edges is large, there may be multiple independent isolating cuts of similar cost for each terminal; an optimal solution will select cuts that share edges, while approximation algorithms may choose cuts with a larger number of edges. We used these observations to generate random instances that are difficult for the multiway cut algorithms. Specifically, we generated three types of random graph instances.

**Simple random graphs:** Edges are added between pairs of randomly selected vertices. After a specific number of edges is added, the largest connected component is output.

**Linear decay random graphs:** These are random graphs where an initial edge density and capacity range is used for edges incident on terminal vertices, and then the edge densities and capacities are linearly decreased as the distance from the terminals increases.

**Exponential decay random graphs:** These graphs are also created by assigning an initial edge density and capacity range to edges incident on terminal vertices, and then exponentially decreasing the densities and capacities as the distance from the terminals increases.

Note that in the linear and exponential decay random graphs edges incident on the terminals have large capacities, ensuring that cuts isolating terminals are not the trivial ones. Edges located far from the terminals are given small capacities, creating a "hot spot" of edges that likely belong to minimum cuts. Furthermore, terminal vertices have the highest number of incident edges, while vertices distant from terminals have the lowest number of incident edges. This edge density gradient encourages optimal solutions that select overlapping cuts through the "hot spot".

Linear and exponential decay random graphs could be used to model practical situations such as strength and availability of wireless signals, traffic congestion around popular destinations, or link capacity and topology in client-server networks.

### 3.8.2 Test Cases

We studied the performance of the approximation algorithms on input networks as described in the previous section, and we studied how graph characteristics such as terminal density ($k/n$), edge capacities, edge density ($m/n$), and number of vertices impact the solution quality, where $n$ is the number of vertices and $m$ is the number of edges. Furthermore, we studied the impact of changing the initial labeling scheme and the value of $\epsilon$ in the local search algorithm.

When $k = 2$ the problem is reduced to the minimum st-cut problem, which can be solved exactly in polynomial time. Additionally, when $k = n$, the problem becomes trivial since every edge must be removed. In order to explore how the number of terminals affects the algorithms, multiple values for $k$ were chosen. For all graph instances where terminals were randomly chosen, $k$ was set to be a fraction of the number of vertices. The fractions used were 3/80, 1/16, 1/8, 1/4, 3/8, and 1/2. For all graph instances this ensured that $k$ was at least 3, but not so large that the problem was trivial.

Assigning larger capacities to edges incident on the terminals led to graph instances whose optimal solutions include edges some distance away from the terminals. We gener-

ated graph instances according to two edge capacity schemes. In the first scheme, edges incident on terminals were assigned rational capacities with values between 30 and 50, while other edges were assigned rational capacities with values between 1 and 25 (in exponential decay random graphs) or between 1 and 45 (in linear decay random graphs). In the second scheme, edges incident on terminals were assigned rational capacities between 1 and 100, while other edges were assigned rational capacites between 1 and 50 (in exponential decay random graphs) or between 1 and 90 (in linear decay random graphs). This second scheme allows edges incident on terminals to have small capacities relative to other edges.

We explored the impact of edge density on the solution quality. Both, the graph instances obtained from the DIMACS competitions and the randomly generated graphs with very large edge densities tended to produce simple instances for which all algorithms produced solutions close to the optimal. Therefore, we concentrated on generating random graphs with a small number of edges. The number of edges chosen for each randomly generated graph were $n$, $2n$, $3n$, $4n$, $5n$, and $6n$, where $n$ is the number of vertices.

We used the following four different initial labeling strategies for the local search algorithm:

**One Each:** Each terminal vertex from 1 to $k$ was assigned a different label, and all of the remaining vertices were assigned the label corresponding to terminal $k$.

**Clumps:** Each terminal vertex from 1 to $k$ was assigned a different label, and the terminals were added to a queue. While the queue had remaining vertices, the first vertex in the queue was removed and the label of that vertex was assigned to all of its unlabeled neighbors, then adding each of these neighbors to the end of the queue.

**Random:** Each terminal vertex from 1 to $k$ was assigned a different label, and the terminals were added to a queue. While the queue had remaining vertices, the queue was randomly shuffled, the first vertex was removed, and the label of that vertex was assigned to all of its unlabeled neighbors, adding each one of them to the queue.

**Isolation Heuristic:** The vertices were assigned labels corresponding to the partitions selected by the isolation heuristic.

These initial labelings provided a variety of starting points that affected the number of iterations needed by the algorithm. We also studied the tradeoff between solution quality and running time produced by the choice for the value of $\epsilon$

### 3.8.3 Results

For randomly generated graphs, 1,000 experiments were performed for each combination of edge density and terminal density for 80 vertex and 160 vertex graphs. Due to their large running times, only 100 experiments were performed on 320 vertex graphs. Graph instances obtained from the DIMACS competitions have a large number of edges, hence only 25 to 100 experiments were performed for each terminal density. We compared the performance of the approximation algorithms according to input network, graph characteristics, and running time. Finally, we anaylsed the performance of the local search algorithm using different initial labelings and values of $\epsilon$.

### Input Networks

Table 3.2 shows a summary of the results[1] for each of the input networks. The value in each entry of the table is calculated by dividing the value of the solution produced by an approximation algorithm by the value of the optimum solution produced by Cplex. The column labeled "Avg" lists the mean of all of the ratios in a test case, and the column labeled "Max" lists the largest ratio produced in a test case. The rows are labeled with the name of the input graph and the number of vertices. For each row, solutions from test cases using different values for $k$, $m$, and edge capacities have been combined.

All of the approximation algorithms computed solutions to the instances from the DIMACS competitions that are very close to the optimal. The exponential decay random graphs caused the highest average ratios. These results show that the isolation heuristic performs the worst, as it computes independent isolating cuts and fails to re-use edges to achieve lower costs. This algorithm does particularily bad on linear and exponential decay random graphs. The algorithms of Calinescu et al., Buchbinder et al., and Sharma and Vondrak, which in the sequel we refer to collectively as the geometric relaxation algorithms, typically compute solutions close to the optimal but occasionally produce solutions near their theoretical worst case performances.

[1]The complete results are split into individual test cases for different values of $n$, $m$, $k$, and edge capacities.

| Test Case | | Isolation Heuristic Avg | Isolation Heuristic Max | Local Search Avg | Local Search Max | Calinescu Avg | Calinescu Max | Buchbinder Avg | Buchbinder Max | Sharma and Vondrak Avg | Sharma and Vondrak Max |
|---|---|---|---|---|---|---|---|---|---|---|---|
| C125 | $n = 125$ | 1.000 | 1.002 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| Brock | $n = 200$ | 1.000 | 1.005 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| Gen | $n = 200$ | 1.000 | 1.005 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| Hamming | $n = 256$ | 1.000 | 1.003 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| Keller | $n = 171$ | 1.000 | 1.005 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| P-Hat | $n = 320$ | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| ST | $n = 80$ | 1.021 | 1.176 | 1.001 | 1.013 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| | $n = 160$ | 1.021 | 1.137 | 1.000 | 1.015 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| | $n = 320$ | 1.023 | 1.145 | 1.000 | 1.005 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| SR | $n = 80$ | 1.031 | 1.507 | 1.001 | 1.133 | 1.000 | 1.038 | 1.000 | 1.034 | 1.000 | 1.024 |
| | $n = 160$ | 1.039 | 1.440 | 1.002 | 1.118 | 1.000 | 1.060 | 1.000 | 1.048 | 1.000 | 1.039 |
| | $n = 320$ | 1.032 | 1.326 | 1.002 | 1.128 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| GL | $n = 80$ | 1.090 | 1.500 | 1.004 | 1.091 | 1.000 | 1.171 | 1.000 | 1.154 | 1.000 | 1.154 |
| | $n = 160$ | 1.106 | 1.600 | 1.010 | 1.133 | 1.000 | 1.068 | 1.000 | 1.089 | 1.000 | 1.093 |
| | $n = 320$ | 1.110 | 1.643 | 1.017 | 1.160 | 1.000 | 1.023 | 1.000 | 1.044 | 1.000 | 1.011 |
| GE | $n = 80$ | 1.120 | 1.500 | 1.006 | 1.087 | 1.000 | 1.292 | 1.000 | 1.292 | 1.000 | 1.196 |
| | $n = 160$ | 1.124 | 1.667 | 1.015 | 1.146 | 1.001 | 1.125 | 1.001 | 1.148 | 1.001 | 1.124 |
| | $n = 320$ | 1.176 | 1.476 | 1.031 | 1.216 | 1.000 | 1.074 | 1.000 | 1.068 | 1.000 | 1.056 |

Table 3.2: Ratios of the solutions computed by approximation algorithms to the optimum, for benchmarks from the DIMACS competitions: Maximum independent set (Brock), maximum clique (Gen, C125), Hamming instances, Keller instances, p-hat instances, and Steiner tree instances (ST). Ratios for the randomly generated instances with simple (SR), linear (GL), and exponential (GE) distributions.

### Graph Characteristics

Table 3.3 shows a sample of results[2] from the exponential decay random distributions, which produced the highest ratios across all of our input networks. Each row is labeled with the number of edges in the test case and rows are grouped according to the number of terminal vertices chosen.

While for the exponential decay random graphs the algorithms had the highest ratios, each multiway cut algorithm only produced these high ratios in a small subset of the test cases. The isolation heuristic had its highest ratios when the value of $k$ was in the range of $0.125n$ to $0.25n$, and when the number of edges was equal to $2n$.

Our local search algorithm and the geometric relaxation algorithms have similar performance. These algorithms produced their highest ratios when the value of $k$ was in the range of $0.0375n$ to $0.125n$, and when the number of edges was between $2n$ and $5n$.

Figure 3.7 shows a sample of plots from the exponential decay random distributions. The local search algorithm has similar performance as the geometric relaxation algorithms. When the number of terminals is small and the number of edges is large, the worst solutions computed by the local search algorithm were close to the average solu-

[2]Many other test cases provided similar results and are not included.

tions. In contrast, the worst solutions computed by the geometric relaxation algorithm were farther from their average solutions.

The local search algorithm performs best when the number of terminals is small. A relabel operation can only expand a single label at a time, but sometimes an improvement exists that is only attainable by changing multiple labels at the same time. Figure 3.5 in Section 3.6 shows an example where the local optimal solution cannot reach the global optimal solution with only single relabel operations. It is possible that networks with a large number of terminals encounter this problem more frequently since there are more possible label configurations for the vertices.

| Test Case | | Isolation Heuristic | | Local Search | | Calinescu | | Buchbinder | | Sharma and Vondrak | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Avg | Max | Avg | Max | Avg | Max | Avg | Max | Avg | Max |
| $k = 3$ | $m = n$ | 1.046 | 1.333 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| | $m = 2n$ | 1.100 | 1.261 | 1.001 | 1.034 | 1.000 | 1.147 | 1.000 | 1.151 | 1.000 | 1.075 |
| | $m = 3n$ | 1.082 | 1.189 | 1.001 | 1.046 | 1.001 | 1.102 | 1.001 | 1.083 | 1.001 | 1.124 |
| | $m = 4n$ | 1.059 | 1.170 | 1.001 | 1.013 | 1.002 | 1.115 | 1.001 | 1.083 | 1.001 | 1.103 |
| | $m = 5n$ | 1.027 | 1.130 | 1.001 | 1.011 | 1.001 | 1.292 | 1.001 | 1.292 | 1.000 | 1.063 |
| | $m = 6n$ | 1.006 | 1.067 | 1.000 | 1.013 | 1.000 | 1.170 | 1.000 | 1.011 | 1.000 | 1.014 |
| $k = 5$ | $m = n$ | 1.128 | 1.500 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| | $m = 2n$ | 1.230 | 1.365 | 1.001 | 1.038 | 1.000 | 1.082 | 1.000 | 1.052 | 1.000 | 1.153 |
| | $m = 3n$ | 1.183 | 1.298 | 1.001 | 1.032 | 1.001 | 1.124 | 1.001 | 1.085 | 1.000 | 1.139 |
| | $m = 4n$ | 1.106 | 1.216 | 1.002 | 1.022 | 1.004 | 1.125 | 1.004 | 1.106 | 1.003 | 1.101 |
| | $m = 5n$ | 1.065 | 1.187 | 1.003 | 1.024 | 1.004 | 1.226 | 1.003 | 1.127 | 1.003 | 1.091 |
| | $m = 6n$ | 1.034 | 1.113 | 1.002 | 1.014 | 1.001 | 1.061 | 1.001 | 1.117 | 1.001 | 1.196 |
| $k = 10$ | $m = n$ | 1.076 | 1.462 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| | $m = 2n$ | 1.320 | 1.460 | 1.000 | 1.039 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| | $m = 3n$ | 1.245 | 1.331 | 1.001 | 1.046 | 1.000 | 1.002 | 1.000 | 1.003 | 1.000 | 1.019 |
| | $m = 4n$ | 1.168 | 1.237 | 1.001 | 1.033 | 1.000 | 1.049 | 1.000 | 1.024 | 1.000 | 1.065 |
| | $m = 5n$ | 1.129 | 1.194 | 1.002 | 1.027 | 1.001 | 1.075 | 1.001 | 1.071 | 1.001 | 1.082 |
| | $m = 6n$ | 1.085 | 1.149 | 1.003 | 1.024 | 1.002 | 1.060 | 1.002 | 1.053 | 1.002 | 1.060 |

Table 3.3: Average and maximum approximation ratios for several test cases on 80 vertex random graphs with exponential decay distributions.

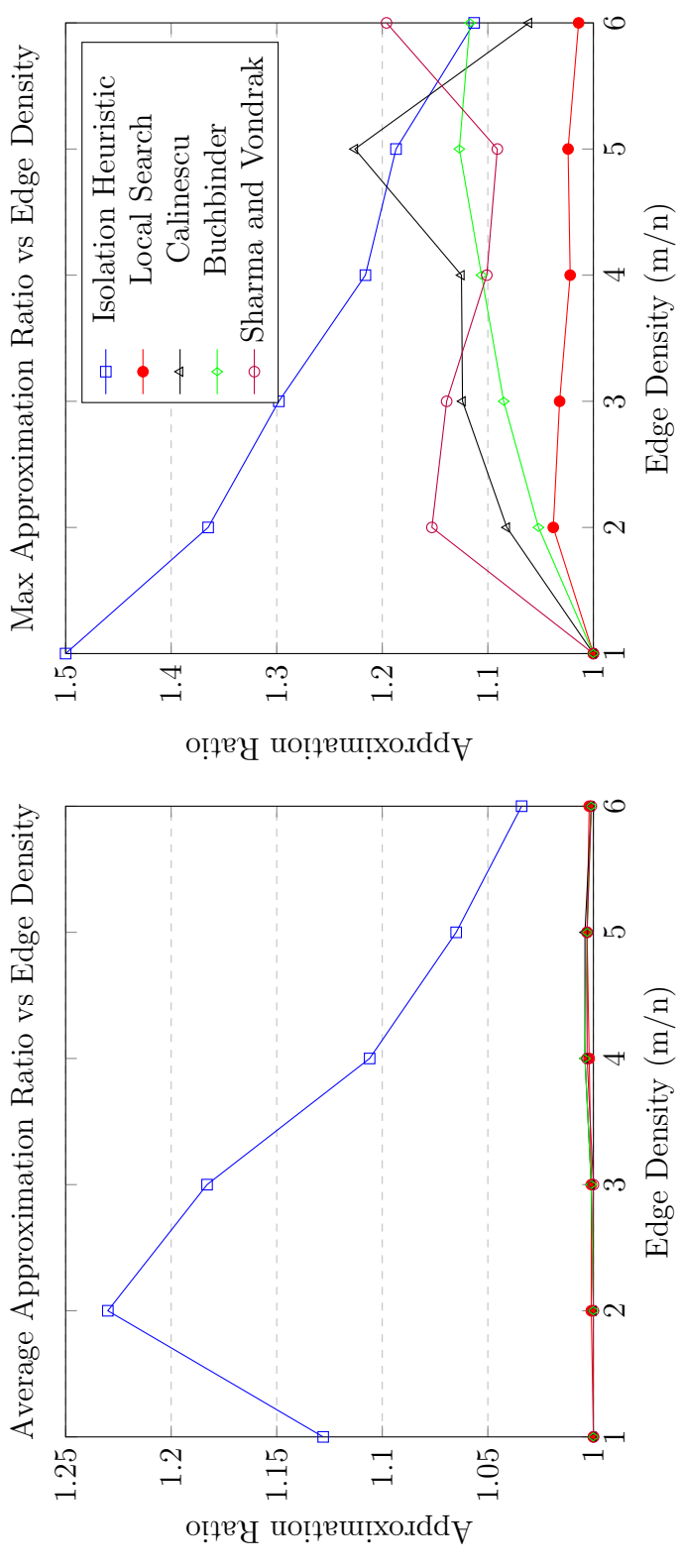


Figure 3.7: Results from the 80 vertex exponential decay graph distribution: Average approximation ratios with 5 terminals (top-left), maximum approximation ratios with 5 terminals (top-right), average approximation ratios with $5n$ edge density (bottom-left), maximum approximation ratios with $5n$ edge density (bottom-right).

### Running Time

A sample of running times for each approximation algorithm is shown in Table 3.4. The isolation heuristic is the fastest algorithm. The running times for the geometric relaxation algorithms include the time needed for Cplex to compute solutions for the linear programs.

The running times of the algorithms of Calinescu et al., Sharma and Vondrak, and Buchbinder et al. are very similar. The running times of these algorithms scale very well with the size of the input graph. The running time of the local search algorithm depends heavily on how the minimum cut is computed and how many relabel operations are performed. When the number of terminals is small, the local search algorithm is faster than the geometric relaxation algorithms.

We improved the running time of the local search algorithm by applying the first relabel operation that improved the cost of the solution instead of computing the relabel operation that had the largest improvement on the cost. We also in each iteration considered first the partition that followed the partition relabeled in the previous iteration. Furthermore, after each partition had been considered at least once, we allowed the local search algorithm to terminate early if $\lceil \frac{k}{2} \rceil$ operations in a row failed to find an improvement. These modifications reduced the number of relabel operations required, but did not have a large negative impact on the solution.

| Test Case | | Isolation Heuristic | Local Search | Calinescu | Buchbinder | Sharma and Vondrak |
|---|---|---|---|---|---|---|
| | | Time (ms) | | | | |
| $n = 80$, $k = 3$ | $m = 2n$ | 1 | 6 | 51 | 51 | 50 |
| | $m = 3n$ | 1 | 14 | 77 | 78 | 77 |
| | $m = 4n$ | 1 | 18 | 91 | 92 | 91 |
| $n = 80$, $k = 5$ | $m = 2n$ | 1 | 15 | 67 | 67 | 67 |
| | $m = 3n$ | 1 | 29 | 81 | 82 | 81 |
| | $m = 4n$ | 1 | 52 | 139 | 141 | 139 |
| $n = 80$, $k = 10$ | $m = 2n$ | 1 | 32 | 93 | 94 | 93 |
| | $m = 3n$ | 1 | 88 | 113 | 114 | 113 |
| | $m = 4n$ | 2 | 126 | 127 | 130 | 127 |
| $n = 160$, $k = 6$ | $m = 2n$ | 2 | 64 | 202 | 203 | 202 |
| | $m = 3n$ | 2 | 160 | 339 | 341 | 339 |
| | $m = 4n$ | 2 | 226 | 534 | 534 | 534 |
| $n = 160$, $k = 10$ | $m = 2n$ | 2 | 137 | 274 | 276 | 274 |
| | $m = 3n$ | 2 | 281 | 336 | 340 | 336 |
| | $m = 4n$ | 3 | 486 | 464 | 470 | 464 |
| $n = 160$, $k = 20$ | $m = 2n$ | 3 | 245 | 352 | 361 | 352 |
| | $m = 3n$ | 4 | 700 | 476 | 483 | 476 |
| | $m = 4n$ | 5 | 970 | 523 | 540 | 523 |

Table 3.4: Running times using 100 experiments for several test cases from the 80 vertex and 160 vertex exponential decay distributions.

### Initial Labeling and Value of $\epsilon$

Figure 3.8 shows how the value of epsilon affects the running times and solution quality for the local search algorithm on the exponential decay graph distribution where $m = 4n$ and $k = 10$. When $(1 - \epsilon/k^2) = 0.9975$, the local search algorithm computes solutions very close to the optimum but requires more time, especially in large graphs. As shown in Table 3.4, we can use $(1-\epsilon/k^2) = 0.99$ to compute high quality solutions quicker than the other algorithms in small graphs. The local search algorithm can run considerably faster by sacrificing solution quality.

The variety of initial labelings for the local search algorithm produced similar solutions when the value of $(1-\epsilon/k^2)$ was between 0.99 and 1.0. When the value of $(1-\epsilon/k^2)$ was less than 0.99, the initial labelings showed differences in performance. The One Each labeling produced the worst solutions; since our random graph structures have many edges and higher capacities on edges incident on terminals, this initial labeling selected many edges with large capacites for the cuts. In contrast, the Clumps initial labeling was closer to the optimal solution and was not significantly affected when the value of $(1 - \epsilon/k^2)$ decreased below 0.99.

When $(1-\epsilon/k^2) = 0.99$ the local search algorithm terminates if it cannot improve its previous best solution by at least 1%. If the initial labeling is very close to the optimal solution, the value of $\epsilon$ has less of an impact; no large improvements remain and the algorithm will quickly terminate. When the initial labeling is further away from the optimum, the value of $\epsilon$ has a large impact.

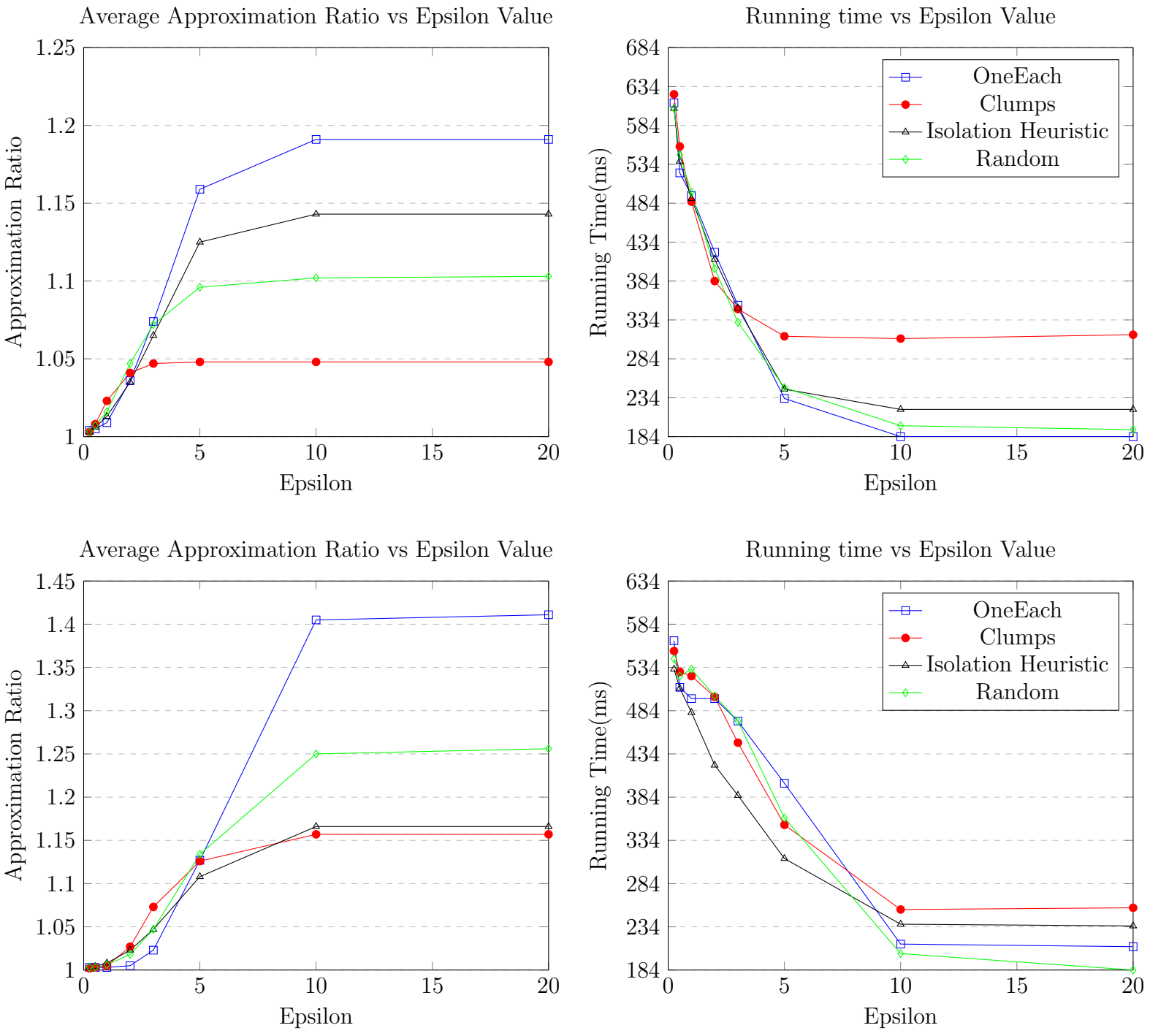


Figure 3.8: Results from the 160 vertex exponential decay graph distribution with $m = 4n$ and $k = 10$. Approximation ratios are compared against the epsilon value using the first edge capacity scheme (top-left) and the second edge capacity scheme (bottom-left). Running times are compared against the epsilon value using the first edge capacity scheme (top-right) and the second edge capacity scheme (bottom-right). The x-axis shows the value of $\epsilon/k^2$; the percentage of improvement to the previous best solution required to continue the iterations of the algorithm.

### 3.8.4 Final Observations

We compared our local search algorithm with four other approximation algorithms for the multiway cut problem. Even though the local search algorithm has the same worst case approximation ratio as the isolation heuristic, its experimental performance is much better. We observed competitive solution quality of the local search algorithm compared to the algorithms of Calinescu et al., Buchbinder et al., and Sharma and Vondrak. On graph with exponential decay random distributions with $k <= 0.125$ and $m >= 2n$ the worst solutions produced by our algorithm were much better than the worst solutions produced by the geometric relaxation algorithms.

On networks with 80 vertices, the local search algorithm computed solutions faster than the geometric relaxation algorithms when $k$ was less than $0.125n$, with the smallest test cases being solved significantly faster by the local search algorithm. On networks with 160 vertices, the local search algorithm computed solutions faster than the geometric relaxation algorithms when $k$ was less than $0.0625n$, but did not scale as well as Cplex, which accounts for the majority of the running time for the other algorithms. Due to the ability to select the value for $\epsilon$, the local search algorithm is more flexible than the other algorithms.

### 3.8.5 Acknowledgements

This work was made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (SHARCNET:www.sharcnet.ca) and Compute/Calcul Canada.

# Bibliography


[1] C.J. Alpert, A. B. Kahng, Recent directions in netlist partitioning: a survey. Integration, the VLSI journal 19, (1995), 1-81.

[2] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, V. Pandit, Local search heuristic for k-median and facility location problem, SIAM Journal on Computing 33, (2004), 544-562.

[3] Y. Boykov, O. Veksler, R. Zabih, Fast approximate energy minimization via graph cuts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 23,* (2001), 1222-1239.

[4] N. Buchbinder, J. Naor, R. Schwartz, Simplex partitioning via exponential clocks and the multiway cut problem, *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, 2013, 535-544.

[5] N. Buchbinder, J. Naor, R. Schwartz, Simplex transformations and the multiway cut problem, *Proceedings of the Twenty-eight Annual ACM-SIAM Symposium on Discrete Algorithms*, 2016.

[6] G. Calinescu, H. Karloff, Y. Rabani, An Improved Approxmation Algorithm for MULTIWAY CUT. Journal of Computer and System Sciences 3, (2000), 564-574.

[7] W.H. Cunningham, L. Tang, Optimal 3-terminal cuts and linear programming, Springer Berlin Heidelberg, (1999), 114-125.

[8] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, M. Yannakakis, The complexity of multiterminal cuts. SIAM Journal on Computing 23, (1994), 864-894.

[9] O. Goldschmidt, D.S. Hochbaum, Polynomial algorithm for the k-cut problem, *In Proceeding of the 29th Annual IEEE Symposium on the Foundations of Computer Science*, Institute of Electrical and Electronics Engineers, New York, (1988), 444-451.

[10] D.R. Karger, P. Klein, C. Stein, M. Thorup, N. E. Young, Rounding algorithms for a geometric embedding of minimum multiway cut, Mathematics of Operations Research 29, (2004), 436-461.

[11] E. L. Lawler, Combinatorial Optimization: Networks and Matroids, Courier Corporation, (1976).

[12] C. H. Lee, M. Kim, C. Park, An efficient k-way graph partitioning algorithm for task allocation in parallel computing systems. *In Proceedings of the First IEEE International Conference on Systems Integration*, New Jersey, (1990), 748-751.

[13] T. Lengauer, Combinatorial algorithms for integrated circuit layout. Springer Science and Business Media, (2012).

[14] J.S. Naor, L. Zosin, A 2-approximation algorithm for the directed multiway cut problem. *In Proceeding of the 38th annual IEEE Symposium on Foundations of Computer Science,* (1997), 548-553.

[15] H. Saran, V. V. Vazirani, Finding $k$ Cuts within Twice the Optimal. SIAM Journal on Computing 24, (1995), 101-108.

[16] A. Sharma, J. Vondrak, Multiway cut, pairwise realizable distribution, and descending thresholds. *In Proceeding of the 46th Annual ACM Symposium on Theory of Computing,* ACM, (2014), 724-733.

[17] H. S. Stone, Multiprocessor scheduling with the aid of network flow algorithms, *IEEE Trans. Software Engineering* ***SE-3****,* (1977), 85-93.

[18] L. Zhao, H. Nagamochi, T. Ibaraki. A greedy splitting algorithm for approximating multiway partition problems. Mathematical Programming 102, (2005), 67-183.

# Chapter 4

# A Local Search Algorithm for the Constrained Max $k$-Cut Problem on Hypergraphs

## 4.1 Introduction

A weighted hypergraph $H = (V, E)$ consist of a set $V$ of nodes, a set $E$ of hyperedges and a function $w$ that assigns a non-negative weight to every edge. A hyperedge $e$ consist of a non-empty set of nodes (called its endpoints). Graphs are special cases of hypergraphs where each hyperedge has exactly two nodes. The size of a hyperedge $e$ is the number of nodes in $e$ and the rank of a hypergraph $H = (V, E)$ is the size of the hyperedge $e \in E$ with the biggest cardinality.

In the max $k$-cut problem on hypergraphs we are given a weighted hypergraph $H = (V, E)$ and an integer $k$, and the goal is to partition $V$ into $k$ non-empty sets in such a way that the sum of the weights of the hyperedges having at least two endpoints in different partitions is maximized.

In the related max multiway cut problem on hypergraphs, besides having a weighted hypergraph $H = (V, E)$ and integer $k$, we are also given a set $T = \{t_1, t_2, \ldots, t_k\} \subseteq V$ of terminals and the goal is to divide $V$ into $k$ partitions so as to maximize the sum of the weights of the hyperedges having at least two endpoints in different partitions and such that each partition has exactly one terminal. Some other related problems are max Steiner $k$-cut, max cut with given sizes of parts [2] and capacitated max $k$-cut [11].

All above problems involve grouping the vertices of a weighted hypergraph $H = (V, E)$ into $k$ non-empty partitions that satisfy some additional set $c$ of constraints and the goal is to maximize the sum of the weights of the hyperedges having at least two endpoints in different partitions. We call this problem the constrained max $k$-cut problem on hypergraphs. For the aforementioned problems the sets $c$ of constraints that their solutions need to satisfy are as follows.

- Max $k$-cut: No additional constraints, just divide $V$ into $k$ disjoint non-empty partitions.

- Max multiway cut: Each partition must include one vertex from a given set $T = \{t_1, t_2, \ldots, t_k\} \subseteq V$ of terminals.
- Max Steiner $k$-cut: Each partition must include at least one vertex from a given set $T = \{t_1, t_2, \ldots, t_l\} \subseteq V$ of terminals, where $l \geq k$. Note that this is a generalization of the max multiway cut problem.
- Capacitated max $k$-cut problem: Given a set $\{s_1, s_2, \ldots, s_k\}$ of sizes, a valid partition $V_1, \ldots, V_k$ of $V$ must satisfy $|V_i| \leq s_i$, for all $1 \leq i \leq k$.
- Max $k$-cut with given sizes of parts: Given a set $\{s_1, s_2, \ldots, s_k\}$ of sizes, a valid partition $V_1, V_2, \ldots, V_k$ of $V$ must satisfy $|V_i| = s_i$, for all $1 \leq i \leq k$. This is a special case of the capacitated max $k$-cut problem.

In this paper we present a general local search algorithm for the constrained max $k$-cut problem on hypergraphs that finds approximate solutions for all aforementioned problems. Our local search algorithm starts with an arbitrary feasible solution for the problem that partitions $V$ into $k$ disjoint sets. The algorithm then tries to improve the current solution by either moving one node from its current partition to another partition or by swapping two nodes from different partitions.

Our algorithm can be modified so it can be used also on the directed max $k$-cut problem on hypergraphs. A directed hypergraph $H = (V, E)$ consist of a set $V$ of nodes and a set $E$ of directed hyperedges. A directed hyperedge is an ordered pair $(t, h)$ formed by two disjoint sets of nodes: $t$ (the tail set) and $h$ (the head set).

Given a directed hypergraph $H = (V, E)$ and a partition $V_1, V_2, \ldots, V_k$ of $V$, the weight of the partition is the total weight of the hyperedges having at least one head in some partition $i$ and at least one of their tails in some partition $j$, where $i > j$. In the directed max $k$-cut problem on hypergraphs, the goal is to find a maximum weight partition $V_1, V_2, \ldots, V_k$ of $V$.

The approximation ratio of our algorithm for max $k$-cut, max multiway cut and max Steiner $k$-cut is $1 - \frac{1}{k}$. For the max $k$-cut problem with given sizes of parts and the capacitated max cut problem our algorithm has approximation ratio $1 - \frac{|V_{max}|}{|V|}$, where $|V_{max}|$ is the size of the largest partition. The approximation ratio of our algorithm for the directed max $k$-cut problem on hypergraphs is $\frac{k-1}{3k-2}$.

**Related Work:** There has been a significant amount of research on max $k$-cut and related problems on graphs. Papadimitriou [20] presented a local search algorithm for the unweighted max cut problem, a special case of the max $k$-cut problem when $k = 2$, and showed that the approximation ratio of his algorithm is $\frac{1}{2}$. This is a simple algorithm that starts with two arbitrary partitions and then repeatedly improves the solution by moving one node to the other partition. Goemans and Williamson [12] introduced a randomized rounding approximation algorithm based on a semidefinite relaxation of the max cut problem with expected approximation ratio 0.8785. They later designed an algorithm for the max 3-cut problem with approximation ratio 0.8360 [13]. An algorithm with the same approximation ratio was presented by Klerk et al. [17].

Vazirani [23] designed a simple greedy $(1 - \frac{1}{k})$-approximation algorithm for the max $k$-cut problem. Frieze and Jerrum [10] generalized the randomized approximation algorithm

of Goemans and Williamson and designed a randomized algorithm for the max $k$-cut problem with expected approximation ratio $1 - \frac{1}{k} + 2\,\frac{ln\,k}{k^2}$. Kann et al. [16] show that no approximation algorithm for the max $k$-cut problem can have approximation ratio better than $1 - \frac{1}{34k}$ unless $P = NP$.

Frieze and Jerrum [10] also designed a randomized algorithm for the max bisection problem, where we have to partition $V$ into two sets of equal size, and showed that the approximation ratio of their algorithm is 0.65. Ye [25] improved on this result by designing an algorithm with approximation ratio 0.699. Later, Halperin and Zwick [14], Feige and Langberg [9], Raghavendra and Tan [21] designed algorithms with approximation ratios 0.7016, 0.7028 and 0.85 respectively, for the same problem. Finally, Austrin et al. [6] improved the approximation ratio to 0.8776.

Currently the best known approximation algorithm for max $k$-Section (in this problem $|V_1| = |V_2| = \ldots = |V_k| = \frac{|V|}{k}$) is by Andersson [4] with approximation ratio $1 - \frac{1}{k} + \Theta(\frac{1}{k^3})$ based on semidefinite programming, generalizing the algorithm in [10] for the max bisection problem.

Liu et al. [18] designed a greedy local search algorithm for the generalized max $k$-multiway cut problem with approximation ratio $1-\frac{1}{k}$. In the generalized max $k$-multiway cut problem besides having a weighted graph $G = (V, E)$ and integer $k$, we are also given $p$ disjoint subsets $U_i$ of $V$ of size $k$. The goal is to divide $V$ into $k$ partitions such that each partition includes exactly one node from $U_i$ for all $1 \leq i \leq p$.

For the max cut problem with given sizes of parts Ageev and Sviridenko [2] desiged a $\frac{1}{2}$-approximation algorithm using pipage rounding. Feige and Langberg [8] designed a semi-definite programming-based algorithm with approximation ratio $\frac{1}{2} + \epsilon$ for the same problem for any $\epsilon > 0$. For the capacitated max $k$-cut problem Wu and Zhu [24] modified the local search algorithm by Gaur et al. [11] and show that the approximation ratio of their algorithm is $\frac{|V_{min}|(k-1)}{2(|V_{max}|-1)+|V_{min}|(k-1)}$, where $|V_{min}|$ and $|V_{max}|$ are sizes of the minimum and the maximum partitions returned by the algorithm. Our algorithm for the capacitated max $k$-cut problem has approximation ratio $1 - \frac{|V_{max}|}{|V|} \geq 1 - \frac{|V_{max}|}{|V_{max}|+|V_{min}|(k-1)} = \frac{|V_{min}|(k-1)}{|V_{max}|+|V_{min}|(k-1)}$. Therefore, our algorithm is better than the algorithm of Wu and Zhu when $\frac{|V_{min}|(k-1)}{|V_{max}|+|V_{min}|(k-1)} > \frac{|V_{min}|(k-1)}{2(|V_{max}|-1)+|V_{min}|(k-1)}$ or in other words when $|V_{max}| \geq 2$. Furthermore, our algorithm works on hypergraphs and not just on graphs.

For the directed max cut problem Goemans and Williamson [12] designed a 0.796-approximation algorithm that uses a semidefinite programming based technique. Feige and Goemens [7] used a similar technique and improved the ratio to 0.859. Also, a $\frac{1}{2}$-approximation algorithm for the max directed cut problem with given sizes of parts was designed by Ageev et al. [1] based on pipage rounding.

For the max cut problem on hypergraphs Andersson and Engebretsen [5] designed a 0.72-approximation algorithm. For the max $k$-cut problem on hypergraphs with given sizes of parts Ageev and Sviridenko [3] designed an approximation algorithm based on pipage rounding with approximation ratio $1-(1-\frac{1}{r})^r-(\frac{1}{r})^r$, where $r$ is the number of nodes in the smallest hyperedge. For the case when all the hyperedges have at least 3 nodes they gave a $(1 - \frac{1}{e})$-approximation algorithm. If we compare our $(1 - \frac{|V_{max}|}{|V|})$-approximation algorithim for the max $k$-cut problem with given sizes of parts on hypergraphs with that

of Ageev and Sviridenko [3], since $1-(1-\frac{1}{r})^r-(\frac{1}{r})^r \leq 0.7$ our algorithm has better approximation ratio when $|V_{max}| < \frac{3}{10}|V|$, where $|V_{max}|$ is the size of the biggest partition.

Zhu and Guo [26] used local search to design a $\frac{k-1}{\Delta+k-1}$-approximation algorithm for the max $k$-cut problem on hypergraphs, where $\Delta = min\{\frac{s(s-1)}{2}, \frac{k(k-1)}{2}\}$ and $s$ is the size of the largest hyperedge. They also gave a local search $(1-\frac{1}{k})$-approximation algorithm for the max $k$- cut problem on graphs. We note that our $(1-\frac{1}{k})$-approximation algorithm for hypergraphs has a much better approximation ratio than that of Zhu and Guo.

## 4.2 The Local Search Algorithm

Given a hypergraph $H=(V,E)$, let $V_1, V_2, \ldots, V_k$ be an arbitrary partition of $V$ into $k$ non-empty sets. We denote a hyperedge $e$ as $(u_1, u_2, \ldots, u_{r_e})$, where $u_1, u_2, \ldots, u_{r_e}$ are the endpoints of $e$. We define $H_i$ to be the set of hyperedges whose endpoints are all in partition $V_i$ and $H_i(u)$ to be the set of hyperedges from $H_i$ incident on $u$:

$$H_i = \{(u_1, u_2, ..., u_{r_e}) \mid u_1, u_2, ..., u_{r_e} \in V_i,\ (u_1, u_2, ..., u_{r_e}) \in E\}, \text{and} \tag{4.1}$$

$$H_i(u) = \{(u_1, u_2, ..., u_{r_e}) \mid u_j = u \text{ for some } 1 \leq j \leq r,\ (u_1, u_2, ..., u_{r_e}) \in H_i\}. \tag{4.2}$$

Let $H_{ij}$ be the set of hyperedges that have one endpoint in $V_i$ and all other endpoints in $V_j$, and let $H_{ij}(u)$ be the set of hyperedges from $H_{ij}$ incident on $u$. Note that in general $H_{ij} \neq H_{ji}$. Our algorithm for the constrained max $k$-cut problem on hypergraphs is described below.

**Algorithm Local Search**$(H, w, c)$
**Input:** Hypergraph $H=(V,E)$, weight function $w: E \to Z^+$, constraints $c$.
**Output:** A partition of the set $V$ satisfying $c$.

1. Start with an arbitrary partition, $V_1$, ..., $V_k$ that satisfies the constraints $c$.

2. If there is a node $u \in V_i$ such that there is a partition $V_l$, $i \neq l$ for which
   $\sum_{e \in H_i(u)} w(e) > \sum_{e \in H_{il}(u)} w(e)$
   and moving $u$ to $V_l$ creates a partition that satisfies the constraints in $c$, then move $u$ from $V_i$ to $V_l$.

3. If there are nodes $u \in V_i$ and $v \in V_l$, $i \neq l$ for which
   $\sum_{e \in H_i(u)} w(e) + \sum_{e \in H_l(v)} w(e) > \sum_{e \in H_{il}(u)} w(e) + \sum_{e \in H_{li}(v)} w(e)$
   and moving $u$ to $V_l$ and $v$ to $V_i$ creates a partition that satisfies the constraints in $c$, then move $u$ to $V_l$ and $v$ to $V_i$.

4. If a node $u$ as specified in Step 2 exists or if nodes $u$, $v$ as specified in Step 3 exist then repeat Steps 2 and 3, otherwise output the partition $V_1, V_2, \ldots, V_k$.

Schaffer and Yannakakis [22] proved that given a weighted graph, the problem of finding a partition of its vertices so the weight of the cut cannot be increased by moving a vertex from one side to the other (same operation as described in Step 2 of our algorithm) is polynomial time local search (PLS)-complete. The class PLS-complete introduced by Johnson et al. [15] is formed by those problems for which a polynomial time local search algorithm for one implies such an algorithm for all of them. Therefore, it is unlikely that our local search algorithm has polynomial running time.

The running time of our local search algorithm is dominated by the time complexity of Step 2 and Step 3 and by the number of times that Step 2 and Step 3 are repeated. Step 2 can be easily implemented to run in $O(k|V|(|V||E|+f(c)))$ time, where the time needed to verify if a partition of $V$ satisfies the constraints in $c$ is $f(c)$, and Step 3 can be implemented to run in $O(|V|^2(|V||E|+f(c)))$ time. The number of iterations of Steps 2 and 3 is at most $\sum_{e\in E} w(e)$ since at each step of the algorithm the weight of the solution increases by at least one unit, but this is not polynomial in the size of the input. Using the result by Orlin et al. [19] we can transform our algorithm into an $\epsilon$-local search algorithm for any $\epsilon > 0$ with approximation ratio $(1-\epsilon)$ times the approximation ratio of the local search algorithm. The running time of the $\epsilon$- local search algorithm is $O(|V|^4(|V||E|+f(c)))$, which is polynomial for any constant value $\epsilon > 0$ when $f(c)$ is polynomial. We note that $f(c)$ is polynomial for all problems mentioned above. In the sequel we will analyze the performance of the local search algorithm knowing that we can modify it to achieve polynomial running time at the expense of a small loss in the quality of the approximation ratio.

## 4.3 Max $k$-Cut, Max Multiway Cut, and Max Steiner $k$-Cut Problems

In this section we analyze the local search algorithm described in the previous section and compute its approximation ratio for the max $k$-cut, the max multiway cut, and the max Steiner $k$-cut problems on hypergraphs.

Let $P = (V_1, V_2, ..., V_k)$ be the partition computed by the local search algorithm. We define $E'$ as the set of hyperedges that have at least two endpoints in different partitions:

$$E' = \{(u_1, u_2, ..., u_{r_e}) \mid \text{partition containing } u_i \neq \text{partition containing } u_j,\ (u_1, u_2, ..., u_{r_e}) \in E\}. \tag{4.3}$$

Then the cost $S$ of the local optimum solution computed by our algorithm is,

$$S = \sum_{e\in E'} w(e). \tag{4.4}$$

Note that the only hyperedges that do not contribute to $S$ are those whose endpoints are all in the same partition. Since $P$ is a local optimal solution, for any nodes $u \in V_i$ and $v \in V_l$, $V_l \neq V_i$, according to the conditions stated in Steps 2 and 3 of the local search algorithm either one or both of the following inequalities hold:

- $$\sum_{e\in H_i(u)} w(e) \leq \sum_{e\in H_{il}(u)} w(e). \tag{4.5}$$

  The above inequality holds if $u$ can be moved to $V_l$ while satisfying the set $c$ of constraints.

- $$\sum_{e\in H_i(u)} w(e) + \sum_{e\in H_l(v)} w(e) \leq \sum_{e\in H_{il}(u)} w(e) + \sum_{e\in H_{li}(v)} w(e). \tag{4.6}$$

  The above inequality holds if $u$ and $v$ can swap partitions while satisfying the set $c$ of constraints.

To make the analysis of the algorithm uniform when applied to any one of the 3 problems considered in this section, for each partition $V_i$, $i = 1, 2, \ldots, k$, we try to choose a node $p_i$ so that inequality (4.6) holds for all pairs of nodes $p_i$, $p_l$, $i \neq l$: We choose (i) $p_i = t_i \in T$ for the max multiway cut problem, (ii) $p_i$ does not exist for the max $k$-cut problem, and (iii) $p_i = t'_i$ for the max Steiner $k$-cut problem, where $t'_i$ is a terminal from $V_i$. Note that inequality (4.5) holds for all nodes $V_i \setminus p_i$, $1 \leq i \leq k$, for all three problems.

Consider partitions $V_l \neq V_i$. If we add Inequality (4.5) for all nodes in $V_i \setminus p_i$ we get,

$$\sum_{u\in V_i\setminus p_i} \sum_{e\in H_i(u)} w(e) \leq \sum_{u\in V_i\setminus p_i} \sum_{e\in H_{il}(u)} w(e). \tag{4.7}$$

Observe that in the term $\sum_{u\in V_i\setminus p_i} \sum_{e\in H_i(u)} w(e)$ the weight of each hyperedge $e \in H_i$ is counted $r_e$ times, except the weight of the hyperedges $e$ incident on the terminals $p_i$ whose weights are counted $r_e - 1$ times. In addition, $\sum_{u\in V_i\setminus p_i} \sum_{e\in H_{il}(u)} w(e)$ includes the weight of all the hyperedges in $H_{il}$ except those incident on terminal $p_i$. Since $r_e \geq 2$ for each hyperedge $e$, we can rewrite Inequality (4.7) as follows,

$$2\sum_{e\in H_i} w(e) - \sum_{e\in H_i(p_i)} w(e) \leq \sum_{e\in H_i} r_e\, w(e) - \sum_{e\in H_i(p_i)} w(e) \leq \sum_{e\in H_{il}} w(e) - \sum_{e\in H_{il}(p_i)} w(e). \tag{4.8}$$

Where $H_i(p_i)$ and $H_{il}(p_i)$ are empty if $p_i$ does not exist. Adding the above inequality over all partitions $V_l \neq V_i$ we get,

$$2(k-1)\sum_{e\in H_i} w(e) - \sum_{\substack{1\leq l\leq k\\ l\neq i}} \sum_{e\in H_i(p_i)} w(e) \leq \sum_{\substack{1\leq l\leq k\\ l\neq i}} \sum_{e\in H_{il}} w(e) - \sum_{\substack{1\leq l\leq k\\ l\neq i}} \sum_{e\in H_{il}(p_i)} w(e). \tag{4.9}$$

Adding this last inequality over all partitions $V_i$ we get,

$$\begin{aligned} 2(k-1) \sum_{1\leq i\leq k} \sum_{e\in H_i} w(e) - \sum_{1\leq i\leq k} \sum_{\substack{1\leq l\leq k\\ l\neq i}} \sum_{e\in H_i(p_i)} w(e) \\ \leq \sum_{1\leq i\leq k} \sum_{\substack{1\leq l\leq k\\ l\neq i}} \sum_{e\in H_{il}} w(e) - \sum_{1\leq i\leq k} \sum_{\substack{1\leq l\leq k\\ l\neq i}} \sum_{e\in H_{il}(p_i)} w(e). \end{aligned} \tag{4.10}$$

Since (4.6) holds for all the nodes $p_i$ then,

$$\sum_{e\in H_i(p_i)} w(e) + \sum_{e\in H_l(p_l)} w(e) \leq \sum_{e\in H_{il}(p_i)} w(e) + \sum_{e\in H_{li}(p_l)} w(e), \text{ for each } 1 \leq i \neq l \leq k. \quad (4.11)$$

We now add up this last inequality over all $i, l = 1, .., k$, $i \neq l$, to get

$$\sum_{1\leq i\leq k} \sum_{\substack{1\leq l\leq k \\ l\neq i}} \left( \sum_{e\in H_i(p_i)} w(e) + \sum_{e\in H_l(p_l)} w(e) \right) \leq \sum_{1\leq i\leq k} \sum_{\substack{1\leq l\leq k \\ l\neq i}} \left( \sum_{e\in H_{il}(p_i)} w(e) + \sum_{e\in H_{li}(p_l)} w(e) \right). \quad (4.12)$$

We can rewrite the above inequality as follows,

$$2 \sum_{1\leq i\leq k} \sum_{\substack{1\leq l\leq k \\ l\neq i}} \sum_{e\in H_i(p_i)} w(e) \leq 2 \sum_{1\leq i\leq k} \sum_{\substack{1\leq l\leq k \\ l\neq i}} \sum_{e\in H_{il}(p_i)} w(e). \quad (4.13)$$

Dividing the above inequality by 2 and adding it to (4.10), we get

$$2(k-1) \sum_{1\leq i\leq k} \sum_{e\in H_i} w(e) \leq \sum_{1\leq i\leq k} \sum_{\substack{1\leq l\leq k \\ l\neq i}} \sum_{e\in H_{il}} w(e). \quad (4.14)$$

Since $\sum_{1\leq i\leq k} \sum_{\substack{1\leq l\leq k \\ l\neq i}} \sum_{e\in H_{il}} w(e) \leq 2S$, then by (4.14)

$$\sum_{1\leq i\leq k} \sum_{e\in H_i} w(e) \leq \frac{1}{k-1} S. \quad (4.15)$$

Since an optimum solution can at most include the weights of all the edges, the cost $O$ of an optimum solution can be bounded by

$$O \leq S + \sum_{1\leq i\leq k} \sum_{e\in H_i} w(e) \leq \left(1 + \frac{1}{k-1}\right) S. \quad (4.16)$$

Therefore,

$$\frac{S}{O} \geq 1 - \frac{1}{k}. \quad (4.17)$$

**THEOREM 8** *There is a $(1 - \frac{1}{k})$-approximation algorithm for the max $k$-cut, max multiway cut, and max Steiner $k$-cut problems on hypergraphs.*

## 4.4 Max Capacitated $k$-Cut Problem and Max $k$-Cut Problem with Given Sizes of Parts

In this section we analyse our local search algorithm for the max capacitated $k$-cut problem and the max $k$-cut problem with given sizes of parts and show that its approximation

ratio is $1 - \frac{|V_{max}|}{|V|}$, where $|V_{max}|$ is the size of the biggest partition returned by the algorithm.

We proceed similarly as in Section 4.3. Since $P = (V_1, V_2, \ldots, V_k)$ is a local optimal solution, for any nodes $u \in V_i$ and $v \in V_l$, $V_l \neq V_i$, either one or both of inequalities (4.5) and (4.6) must hold. Observe that in the max $k$-cut problem with given sizes of parts only swaps are allowed, therefore only inequality (4.6) is true for all the nodes. On the other hand, in the capacitated max $k$-cut problem the condition in Step 2 of the algorithm is true for a node $u \in V_i$ only if there is a partition $V_l \neq V_i$ of size $|V_l| < s_l$ and such that $\sum_{e \in H_i(u)} w(e) > \sum_{e \in H_{il}(u)} w(e)$. Since swaps are allowed for all pairs of nodes in the capacitated max $k$-cut problem Inequality (4.6) is true for all of them; hence in the analysis we will only use this inequality.

Adding Inequality (4.6) for all $u \in V_i$ we get,

$$\sum_{e \in H_i} r_e w(e) + |V_i| \sum_{e \in H_l(v)} w(e) \leq \sum_{e \in H_{il}} w(e) + |V_i| \sum_{e \in H_{li}(v)} w(e). \tag{4.18}$$

Notice that the first term in the left side of this inequality is $\sum_{e \in H_i} r_e w(e)$ because each hyperedge $e$ in $H_i$ is counted exactly $r_e$ times in $\sum_{u \in V_i} \sum_{e \in H_i(u)} w(e)$ and the first term in the right side of the inequality is $\sum_{e \in H_{il}} w(e)$ since each hyperedge in $H_{il}$ is counted exactly one time in $\sum_{u \in V_i} \sum_{e \in H_{il}(u)} w(e)$. Next, we sum Inequality (4.18) for all $v \in V_l$ to get

$$|V_l| \sum_{e \in H_i} r_e w(e) + |V_i| \sum_{e \in H_l} r_e w(e) \leq |V_l| \sum_{e \in H_{il}} w(e) + |V_i| \sum_{e \in H_{li}} w(e). \tag{4.19}$$

Since $r_e \geq 2$ for each hyperedge then,

$$2|V_l| \sum_{e \in H_i} w(e) + 2|V_i| \sum_{e \in H_l} w(e) \leq |V_l| \sum_{e \in H_i} r_e w(e) + |V_i| \sum_{e \in H_l} r_e w(e) \leq |V_l| \sum_{e \in H_{il}} w(e) + |V_i| \sum_{e \in H_{li}} w(e). \tag{4.20}$$

We sum this inequality for all $i, l = 1, 2, \ldots, k$, $i \neq l$:

$$\sum_{1 \leq i \leq k} \sum_{\substack{1 \leq l \leq k \\ i \neq l}} 2(|V_l| \sum_{e \in H_i} w(e) + |V_i| \sum_{e \in H_l} w(e)) \leq \sum_{1 \leq i \leq k} \sum_{\substack{1 \leq l \leq k \\ i \neq l}} (|V_l| \sum_{e \in H_{il}} w(e) + |V_i| \sum_{e \in H_{li}} w(e)). \tag{4.21}$$

The left side of the above inequality can be simplified as follows,

$$\begin{aligned} \sum_{1 \leq i \leq k} \sum_{\substack{1 \leq l \leq k \\ i \neq l}} 2(|V_l| \sum_{e \in H_i} w(e) + |V_i| \sum_{e \in H_l} w(e)) &= 2 \sum_{1 \leq i \leq k} \sum_{e \in H_i} w(e) \sum_{\substack{1 \leq l \leq k \\ i \neq l}} |V_l| + \\ 2 \sum_{1 \leq l \leq k} \sum_{e \in H_l} w(e) \sum_{\substack{1 \leq i \leq k \\ i \neq l}} |V_i| &= 2 \sum_{1 \leq i \leq k} (|V| - |V_i|) \sum_{e \in H_i} w(e) + \\ 2 \sum_{1 \leq l \leq k} (|V| - |V_l|) \sum_{e \in H_l} w(e) &= 4 \sum_{1 \leq i \leq k} (|V| - |V_i|) \sum_{e \in H_i} w(e). \end{aligned}$$

Similarly, the right side of Inequality (4.21) can be simplified as follows,

$$\sum_{1\le i\le k}\sum_{\substack{1\le l\le k\\ i\neq l}}(|V_l|\sum_{e\in H_{il}}w(e)+|V_i|\sum_{e\in H_{li}}w(e))=\sum_{1\le i\le k}\sum_{\substack{1\le l\le k\\ i\neq l}}|V_l|\sum_{e\in H_{il}}w(e)+$$
$$\sum_{1\le l\le k}\sum_{\substack{1\le i\le k\\ i\neq l}}|V_i|\sum_{e\in H_{li}}w(e)=2\sum_{1\le i\le k}\sum_{\substack{1\le l\le k\\ i\neq l}}|V_i|\sum_{e\in H_{il}}w(e).$$

Therefore, we can re-write Inequality (4.21) as follows,

$$2\sum_{1\le i\le k}(|V|-|V_i|)\sum_{e\in H_i}w(e)\le\sum_{1\le i\le k}\sum_{\substack{1\le l\le k\\ i\neq l}}|V_i|\sum_{e\in H_{il}}w(e). \tag{4.22}$$

Let $|V_{max}|= max\{|V_i|, i=1,2,\ldots,k\}$, then

$$2(|V|-|V_{max}|)\sum_{1\le i\le k}\sum_{e\in H_i}w(e)\le|V_{max}|\sum_{1\le i\le k}\sum_{\substack{1\le l\le k\\ i\neq l}}\sum_{e\in H_{il}}w(e)\le 2|V_{max}|\ S \tag{4.23}$$

Therefore,

$$\sum_{1\le i\le k}\sum_{e\in H_i}w(e)\le\frac{|V_{max}|}{|V|-|V_{max}|}S. \tag{4.24}$$

Since,

$$O\le S+\sum_{1\le i\le k}\sum_{e\in H_i}w(e)\le(1+\frac{|V_{max}|}{|V|-|V_{max}|})S, \tag{4.25}$$

then,

$$\frac{S}{O}\ge\frac{1}{1+\frac{|V_{max}|}{|V|-|V_{max}|}}=1-\frac{|V_{max}|}{|V|} \tag{4.26}$$

**THEOREM 9** *There is a $(1-\frac{|V_{max}|}{|V|})$-approximation algorithm for the max capacitated $k$-cut problem and max $k$-cut problem with given sizes of parts on hypergraphs.*

**Corollary 4.4.1** *There is a $\frac{1-|V_{max}|}{2|V|-|V_{max}|}$-approximation algorithm for the max capacitated $k$-cut problem and the max $k$-cut problem with given sizes of parts restricted to hypergraphs where every hyperedge has at least 3 endpoints.*

**Proof** Note that if every hyperedge has at least three endpoints then inequality (4.23) becomes $2(|V|-|V_{max}|)\sum_{1\le i\le k}\sum_{e\in H_i}\le|V_{max}|S$ and thus in this case $\frac{S}{O}\ge 1-\frac{|V_{max}|}{2|V|-|V_{max}|}$.

## 4.5 Directed Max $k$-Cut Problem

A directed hypergraph $H = (V, E)$ consist of set $V$ of nodes and set $E$ of hyperedges. Each hyperedge $e = (u_1, u_2, \ldots, u_{r_e}) \in E$ has a set $t_e$ of tails and, a set $h_e$ of heads and a weight $w(e)$. We call a hyperedges $e$, a *B-arc* if $e$ has only one head $h_e$ and a *F-arc* if $e$ has only one tail $t_e$. A *BF-hypergraph* is a directed hypergraph in which all the hyperedges are B-arcs or F-arcs. In this section we deal with BF-hypergraphs, so in the sequel hypergraph means BF-hypergraph.

Given a directed hypergraph $H = (V, E)$ and a partition $V_1, V_2, \ldots, V_k$ of $V$, the weight of the partition is the total weight of the hyperedges having at least one head in some partition $i$ and at least one of their tails in some partition $j$, where $i > j$. In the directed max $k$-cut problem on hypergraphs, the goal is to find a maximum weight partition $P = V_1, V_2, \ldots, V_k$ of $V$.

In Figure 4.1 a hypergraph $H = (V, E)$ with 8 vertices and 5 hyperedges is shown. The sets of tails and heads for each hyperedge are as follows, $t_{e_1} = \{v_1\}$, $h_{e_1} = \{v_2\}$, $t_{e_2} = \{v_4\}$, $h_{e_2} = \{v_2, v_3\}$, $t_{e_3} = \{v_1\}$, $h_{e_3} = \{v_5\}$, $t_{e_4} = \{v_4\}$, $h_{e_4} = \{v_6, v_7, v_8\}$, $t_{e_5} = \{v_5\}$ and $h_{e_5} = \{v_4, v_8\}$. Let 3, 4, 1, 5, 1 be the weights of hyperedges $e_1$, $e_2$, $e_3$, $e_4$ and $e_5$ respectively. Consider partition $P = V_1, V_2, V_3$. The weight of this partition is 1+5+1=7.

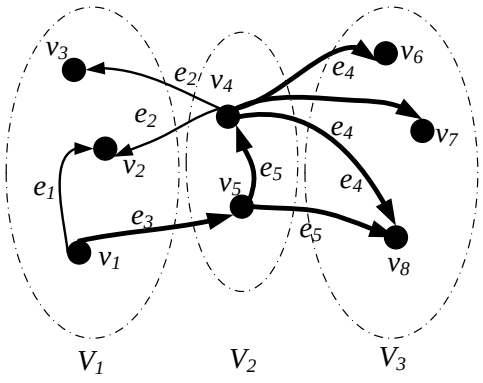


Figure 4.1: Example of a directed Hypergraph.

Given a hypergraph $H = (V, E)$, and a partition $P = V_1, V_2, \ldots, V_k$ of $V$ we define sets $H_i$, $H_i(u)$, $T_i(u)$, as follows,

$$H_i = \{(u_1, u_2, \ldots, u_{r_e}) \mid u_1, u_2, \ldots, u_{r_e} \in V_i,\ (u_1, u_2, \ldots, u_{r_e}) \in E\},$$

$$H_i(u) = \{e = (u_1, u_2, \ldots, u_{r_e}) \mid (u_1, u_2, \ldots, u_{r_e}) \in H_i,\ u \in h_e\},$$

$$T_i(u) = \{e = (u_1, u_2, \ldots, u_{r_e}) \mid\ (u_1, u_2, \ldots, u_{r_e}) \in H_i,\ u \in t_e\}.$$

We define additional sets of hyperedges $T_{ij}$ and $H_{ij}$ as follows.

- $T_{ij}$, $i < j$, is a set of B-arcs and F-arcs that contribute to the weight of the partition $P$ such that if we move one of the tails of any of these hyperedges from $V_i$ to $V_j$ then that hyperedge will no longer contribute to the weight of the partition. The hyperedges of $T_{ij}$ have the following properties:

(i) each B-arc $e$ in $T_{ij}$ has exactly one tail in $V_i$ and every other tail in $\bigcup_{j\leq q\leq k} V_q$, and its head is in $V_j$,

(ii) each F-arc $e$ in $T_{ij}$ has its tail in $V_i$, at least one head in $V_j$ and no head in $\left(\bigcup_{j<q\leq k} V_q\right)$.

Let $T_{ij}(u)$, $u \in V_i$, be the set of hyperedges $e$ from $T_{ij}$ for which $u \in t_e$.

- $H_{ij}$, $i > j$, is a set of B-arcs and F-arcs that contribute to the weight of partition $P$ such that if we move one of the heads of any of these hyperedges from partition $V_i$ to partition $V_j$ then that hyperedge will no longer contribute to the weight of $P$. The hyperedges of $H_{ij}$ have the following properties:

  (i) each B-arc $e$ in $H_{ij}$ has its head in $V_i$, no tail in $\bigcup_{1\leq q<j} V_q$, and at least one tail in $V_j$,

  (ii) each F-arc $e$ in $H_{ij}$ has exactly one head in $V_i$ and all other heads in $\bigcup_{1\leq q\leq j} V_q$, and its tail in $V_j$.

  Let $H_{ij}(u)$, $u \in V_i$, be the set of hyperedges $e$ from $H_{ij}$, where $u \in h_e$.

Our algorithm for the directed max $k$-cut problem is described below.

**Algorithm Max DICUT** $(H, w)$
**Input:** Directed hypergraph $H = (V, E)$, hyperedge weight function $w : E \to Z^+$.
**Output:** A partition of the set $V$.

1. Start with an arbitrary partition, $V_1$, ..., $V_k$, where $V_i \neq \emptyset$ for $i = 1, 2, \ldots, k$.

2. If there is a node $u \in V_i$ and a partition $V_l$, $i < l$, such that

$$\sum_{e\in H_i(u)} w(e) > \sum_{i<j\leq l} \sum_{e\in T_{ij}(u)} w(e),$$

   then move $u$ from $V_i$ to $V_l$.

3. If there is a node $u \in V_i$ and a partition $V_l$, $i > l$, such that

$$\sum_{e\in T_i(u)} w(e) > \sum_{l\leq j<i} \sum_{e\in H_{ij}(u)} w(e),$$

   then move $u$ from $V_i$ to $V_l$.

4. If a node $u$ as specified in Step 2 or Step 3 exists then repeat Step 2 and Step 3. Otherwise, compare the cost of the solution induced by the ordered partition $P = V_1, V_2, \ldots, V_k$ and the cost of the solution induced by the reverse partition $P_r = V_k, V_{k-1}, \ldots, V_1$ and return the partition with the bigger cost.

Using the local search property specified in Step 2 of the algorithm, for each node $u \in V_i$, $i, l \in \{1, 2, \ldots, k\}$ and $i < l$ we have,

$$\sum_{e \in H_i(u)} w(e) \leq \sum_{e \in T_{ij}(u)} w(e). \tag{4.27}$$

Adding up Inequality (4.27) for all nodes in $V_i$ we get,

$$\sum_{u \in V_i} \sum_{e \in H_i(u)} w(e) \leq \sum_{u \in V_i} \sum_{i < j \leq l} \sum_{e \in T_{ij}(u)} w(e). \tag{4.28}$$

Observe that each hyperedge $e$ in the term $\sum_{u \in V_i} \sum_{e \in H_i(u)} w(e)$ is counted $|h_e|$, times therefore $\sum_{e \in H_i} w(e) \leq \sum_{e \in H_i} |h_e| w(e) = \sum_{u \in V_i} \sum_{e \in H_i(u)} w(e)$. In the term $\sum_{u \in V_i} \sum_{i < j \leq l} \sum_{e \in T_{ij}(u)} w(e)$ each hyperedge $e$ is counted once because in this expression $e$ is counted only when $u \in t_e \cap V_i$ and from the definition of $T_{ij}(u)$ we know that $u$ must be a tail of $e$, at least one head of $e$ must be in $V_j$ and no head of $e$ can be in $V_q$ for $j < q \leq k$. Therefore, Inequality (4.28) can be simplified as follows,

$$\sum_{e \in H_i} w(e) \leq \sum_{i < j \leq l} \sum_{e \in T_{ij}} w(e). \tag{4.29}$$

Adding (4.29) over all $1 \leq i < l \leq k$, we get

$$\sum_{1 \leq i \leq k} \sum_{i < l \leq k} \sum_{e \in H_i} w(e) \leq \sum_{1 \leq i \leq k} \sum_{i < l \leq k} \sum_{i < j \leq l} \sum_{e \in T_{ij}} w(e). \tag{4.30}$$

Similarly, using the local search property specified in Step 3 of the algorithm, for each node $u \in V_i$, $i, l \in \{1, 2, \ldots, k\}$ and $l < i$, we have,

$$\sum_{e \in T_i(u)} w(e) \leq \sum_{l \leq j < i} \sum_{e \in H_{ij}(u)} w(e). \tag{4.31}$$

Adding up Inequality (4.31) for all nodes in $V_i$ we get,

$$\sum_{u \in V_i} \sum_{e \in T_i(u)} w(e) \leq \sum_{u \in V_i} \sum_{l \leq j < i} \sum_{e \in H_{ij}(u)} w(e). \tag{4.32}$$

Observe that by a similar argument as above $\sum_{e \in T_i} w(e) \leq \sum_{u \in V_i} \sum_{e \in T_i(u)} w(e)$. Also, in the term $\sum_{u \in V_i} \sum_{l \leq j < i} \sum_{e \in H_{ij}(u)} w(e)$ in the right side of (4.32) each hyperedge $e$ is counted once. To see this consider the following two cases: If $e$ is a B-arc then $e$ has its head in $V_i$, at least one tail in $V_j$ and no tail in $\bigcup_{1 \leq q < j} V_q$; hence, in the right side of (4.32) $e$ is counted only once when $j$ is the smallest index of a partition containing a tail of $e$. If $e$ is an F-arc then it has exactly one head in $V_i$, its tail in $V_j$ and all other heads in $\cup_{1 \leq q \leq j} V_q$; therefore, in the right side of (4.32) $e$ is only counted once when $j$ is the index of the partition containing the tail of $e$.

Therefore, Inequality (4.32) can be simplified as follows,

$$\sum_{e \in H_i} w(e) \leq \sum_{l \leq j < i} \sum_{e \in H_{ij}} w(e). \tag{4.33}$$

Adding Inequality (4.33) over all $1 \leq l < i \leq k$, we get

$$\sum_{1\leq i\leq k}\sum_{1\leq l<i}\sum_{e\in H_i} w(e) \leq \sum_{1\leq i\leq k}\sum_{1\leq l<i}\sum_{l\leq j<i}\sum_{e\in H_{ij}} w(e). \tag{4.34}$$

Now we add inequalities (4.30) and (4.34):

$$\sum_{1\leq i\leq k}\sum_{\substack{1\leq l\leq k\\ i\neq l}}\sum_{e\in H_i} w(e) \leq \sum_{1\leq i\leq k}\sum_{i<l\leq k}\sum_{i<j\leq l}\sum_{e\in T_{ij}} w(e) + \sum_{1\leq i\leq k}\sum_{1\leq l<i}\sum_{l\leq j<i}\sum_{e\in H_{ij}} w(e). \tag{4.35}$$

Each term $\sum_{e\in T_{ij}} w(e)$ is counted $k-j+1$ times in $\sum_{1\leq i\leq k}\sum_{i<l\leq k}\sum_{i<j\leq l}\sum_{e\in T_{ij}} w(e)$ because for each pair $i$, $j$, $i<j$, the value of $l$ must be such that $j \leq l$ and $l \leq k$; since there are $k-j+1$ such values, the term $\sum_{e\in T_{ij}}$ appears $k-j+1$ times. Similarly, the term $\sum_{e\in H_{ij}} w(e)$, $1 \leq j < i \leq k$, is counted $j$ times in $\sum_{1\leq i\leq k}\sum_{1\leq l<i}\sum_{l\leq j<i}\sum_{e\in H_{ij}} w(e)$, because for each pair $i$, $j$, $i<j$, the value of $l$ must be such that $l \geq 1$ and $l \leq j$; since there are $j$ such values, the term $\sum_{e\in H_{ij}} w(e)$ appears $j$ times. Therefore, we can rewrite the right hand side of (4.35) as follows,

$$\begin{aligned}&\sum_{1\leq i\leq k}\sum_{i<l\leq k}\sum_{i<j\leq l}\sum_{e\in T_{ij}} w(e) + \sum_{1\leq i\leq k}\sum_{1\leq l<i}\sum_{l\leq j<i}\sum_{e\in H_{ij}} w(e) = \\ &\sum_{1\leq i\leq k}\sum_{i<j\leq k}(k-j+1)\sum_{e\in T_{ij}} w(e) + \sum_{1\leq i\leq k}\sum_{1\leq j<i} j\sum_{e\in H_{ij}} w(e).\end{aligned} \tag{4.36}$$

Observe that in the term $\sum_{1\leq i\leq k}\sum_{1\leq j<i} j\sum_{e\in H_{ij}} w(e)$ if we replace $i$ with $j$ and $j$ with $i$ then we get,

$$\sum_{1\leq i\leq k}\sum_{1\leq j<i} j\sum_{e\in H_{ij}} w(e) = \sum_{1\leq j\leq k}\sum_{1\leq i<j} i\sum_{e\in H_{ji}} w(e). \tag{4.37}$$

Note that in the term $\sum_{1\leq j\leq k}\sum_{1\leq i<j} i\sum_{e\in H_{ji}} w(e)$, $i$ can get values from 1 to $k-1$ and $j$ can get values from $i+1$ to $k$, therefore,

$$\sum_{1\leq j\leq k}\sum_{1\leq i<j} i\sum_{e\in H_{ji}} w(e) = \sum_{1\leq i<k}\sum_{i<j\leq k} i\sum_{e\in H_{ji}} w(e) = \sum_{1\leq i\leq k}\sum_{i<j\leq k} i\sum_{e\in H_{ji}} w(e). \tag{4.38}$$

The second equality in (4.38) is true since, if $i=k$ there is no value $j$ such that $i<j\leq k$. Let $E_{ij} = T_{ij} \cup H_{ji}$, for each $i<j$. Using (4.37) and (4.38) in the right hand side of (4.36) we get,

$$\begin{aligned}&\sum_{1\leq i\leq k}\sum_{i<j\leq k}(k-j+1)\sum_{e\in T_{ij}} w(e) + \sum_{1\leq i\leq k}\sum_{1\leq j<i} j\sum_{e\in H_{ij}} w(e) = \\ &\sum_{1\leq i\leq k}\sum_{i<j\leq k}(k-j+1)\sum_{e\in T_{ij}} w(e) + \sum_{1\leq i\leq k}\sum_{i<j\leq k} i\sum_{e\in H_{ji}} w(e) \leq \\ &\sum_{1\leq i\leq k}\sum_{i<j\leq k}(k-j+1)\sum_{e\in E_{ij}} w(e) + \sum_{1\leq i\leq k}\sum_{i<j\leq k} i\sum_{e\in E_{ij}} w(e) = \\ &\sum_{1\leq i\leq k}\sum_{i<j\leq k}(k+i-j+1)\sum_{e\in E_{ij}} w(e) \leq k\sum_{1\leq i\leq k}\sum_{i<j\leq k}\sum_{e\in E_{ij}} w(e).\end{aligned} \tag{4.39}$$

The last inequality holds because $i < j$. Now we show that all sets $E_{ij}$, for all $1 \leq i < j \leq k$, are disjoint. Suppose that there are sets $E_{ij}$ and $E_{lq}$, $E_{ij} \neq E_{lq}$, $1 \leq i < j \leq k$ and $1 \leq l < q \leq k$, such that $E_{ij} \cap E_{lq} \neq \emptyset$.

- Let $E_{ij}$ and $E_{lq}$ share a B-arc $e$. Recall that by the definition of B-arcs, $e$ has one head. Without loss of generality assume $l < i$. Since $E_{ij} = T_{ij} \cup H_{ji}$, by the definition of $T_{ij}$ and $H_{ji}$ if $e \in E_{ij}$ then $e$ has its head in $V_j$, at least one tail in $V_i$, and no tails in $\bigcup_{1 \leq t < i} V_t$ (observe that if $e \in T_{ij}$ then $e$ has exactly one tail in $V_i$ and all other tails are in $\bigcup_{j \leq t \leq k} V_t$, and since $i < j$ then there is no tail in $\bigcup_{1 \leq t < i} V_t$). Similarly if $e \in E_{lq}$, then $e$ should have its head in $V_q$, and since $e$ has only one head then it must be that $V_j = V_q$; furthermore $e$ has at least one tail in $V_l$, however since $l < i$ this contradicts the fact that $e$ has no tails in $\bigcup_{1 \leq t < i} V_t$.
- Now suppose that $E_{ij}$ and $E_{lq}$ share a F-arc $e$. Recall that F-arcs have only one tail. Without loss of generality assume that $j < q$. Since $E_{ij} = T_{ij} \cup H_{ji}$, by the definition of $T_{ij}$ and $H_{ji}$ if $e \in E_{ij}$ then it has its tail in $V_i$, at least one head in $V_j$ and no head in $\bigcup_{j < t \leq k} V_t$. Similarly if $e \in E_{lq}$, $e$ has its tail in $V_l$ and since $e$ has only one tail then $V_i = V_l$; moreover $e$ has at least one head in $V_q$, however since $j < q$ and $e$ cannot have any heads in $\bigcup_{j < t \leq k} V_t$ this is a contradiction.

Let $A_{ij}$, $i < j$ be the set of hyperedges that have at least one tail in $V_i$ and at least one head in $V_j$; note that $E_{ij} \subseteq A_{ij}$ for each $i < j$. The weight of the local optimal partition $P$ is the weight of all the hyperedges in $\bigcup_{1 \leq i < j \leq k} A_{ij}$, and since $E_{ij} \subseteq A_{ij}$ then $\bigcup_{1 \leq i < j \leq k} E_{ij} \subseteq \bigcup_{1 \leq i < j \leq k} A_{ij}$. Given a set $C$ of hyperedges, let $w(C)$ denote the weight of the hyperedges of $C$. Then $w(\bigcup_{1 \leq i < j \leq k} E_{ij}) \leq w(\bigcup_{1 \leq i < j \leq k} A_{ij}) = S$, where $S$ is the weight of the local optimal solution. Since all the sets $E_{ij}$, $1 \leq i, j \leq k$, are disjoint then $w(\bigcup_{1 \leq i < j \leq k} E_{ij}) = \sum_{1 \leq i \leq k} \sum_{i < j \leq k} \sum_{e \in E_{ij}} w(e)$, and so the right side of the last inequality in (4.39) can be bounded as follows,

$$k \sum_{1 \leq i \leq k} \sum_{i < j \leq k} \sum_{e \in E_{ij}} w(e) \leq kS. \tag{4.40}$$

We can simplify the left side of Inequality (4.35):

$$\sum_{1 \leq i \leq k} \sum_{\substack{1 \leq l \leq k \\ i \neq l}} \sum_{e \in H_i} w(e) = (k-1) \sum_{1 \leq i \leq k} \sum_{e \in H_i} w(e) \tag{4.41}$$

Therefore, by inequalities (4.35), (4.36), (4.39), (4.40) and (4.41) we have,

$$(k-1) \sum_{1 \leq i \leq k} \sum_{e \in H_i} w(e) \leq kS, \text{ or } \sum_{1 \leq i \leq k} \sum_{e \in H_i} w(e) \leq \frac{k}{k-1} S. \tag{4.42}$$

Let $B$ be the set of hyperedges in $E - S_L - \bigcup_{1 \leq i \leq k} \bigcup_{e \in H_i} e$, where $S_L$ is the set of hyperedges that contribute to the weight of the local optimal solution. Let $S_r$ be the set of hyperedges that contribute to the weight of the reverse partition $P_r = V_k, V_{k-1}, \ldots, V_1$ as described in Step 4 of the algorithm. Note that because of the last step of the algorithm,

$S \geq w(S_r)$, and since $w(B) \leq w(S_r)$ then $w(B) \leq S$. Let $O$ be the weight of an optimal solution. Adding $w(B) + S$ to left side of inequality (4.42) and $2S$ to the right side we get,

$$O \leq w(B) + S + \sum_{1 \leq i \leq k} \sum_{e \in H_i} w(e) \leq 2S + \frac{k}{k-1} S. \tag{4.43}$$

Therefore,

$$\frac{k-1}{3k-2} \leq \frac{S}{O}. \tag{4.44}$$

**THEOREM 10** *There is a $\frac{k-1}{3k-2}$-approximation algorithm for the directed max $k$-cut problem on hypergraphs.*

# Bibliography


[1] A. A. Ageev, R. Hassin, and M. I. Sviridenko. A 0.5-Approximation algorithm for MAX DICUT with given sizes of parts. SIAM Journal on Discrete Mathematics, 14(2): 246-255, 2001.

[2] A. A. Ageev and M. I. Sviridenko. Approximation algorithms for maximum coverage and max cut with given sizes of parts. In Integer Programming and Combinatorial Optimization, volume 1610 of Lecture Notes in Computer Science, pages 17-30. Springer, 1999.

[3] A. A. Ageev and M. I. Sviridenko. An approximation algorithm for hypergraph max k-Cut with given sizes of parts. In Algorithms - ESA 2000, volume 1879 of Lecture Notes in Computer Science, pages 32-41. Springer, 2000.

[4] G. Andersson. An approximation algorithm for max p-Section. In STACS 99, volume 1563 of Lecture Notes in Computer Science, pages 237-247. Springer, 1999.

[5] G. Andersson and L. Engebretsen. Better approximation algorithms for SET SPLITTING and NOT-ALL-EQUAL SAT. Information Processing Letters, 65(6): 305-311, 1998.

[6] P. Austrin, S. Benabbas, and K. Georgiou. Better balance by being biased: A 0.8776-approximation for max bisection. In SODA, pages 277-294, 2013.

[7] U. Feige and M.X. Goemans, Approximating the value of two prover proof systems, with applications to MAX 2SAT and MAX DICUT, Proceedings of the Third Israel Symposium on Theory of Computing and Systems, Tel Aviv, Israel, 182–189, 1995.

[8] U. Feige and M. Langberg. Approximation algorithms for maximization problems arising in graph partitioning. Journal of Algorithms, 41(2): 174-211, 2001.

[9] U. Feige and M. Langberg. The RPR2 Rounding Technique For Semidefinite Programs. Journal of Algorithms, 60(1): 1-23, 2006.

[10] A. Frieze and M. Jerrum. Improved approximation algorithms for MAX-k-CUT and MAX BISECTION. Algorithmica, 18(1): 67-81, 1997.

[11] D. R. Gaur, R. Krishnamurti, R. Kohli. The capacitated max k-cut problem. Mathematical Programming, 115(1): 65-72, 2008.

[12] M. X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of the ACM, 42(6): 1115-1145, 1995.

[13] M. X. Goemans and D. P. Williamson. Approximation algorithms for Max-3-Cut and other problems via complex semidefinite programming. Journal of Computer and System Sciences, 68(2): 442-470, 2004.

[14] E. Halperin and U. Zwick. A unified framework for obtaining improved approximation algorithms for maximum graph bisection problems. Random Struct. Algorithms, 20(3): 382-402, 2002.

[15] D. S. Johnson, C. H. Papadimitriou, M. Yannakakis. How easy is local search? Journal of Computer and System Sciences, 37: 79-100, 1988.

[16] V. Kann, S. Khanna, J. Lagergren, and A. Panconesi. On the hardness of approximating MAX k-CUT and its dual. Chicago Journal of Theoretical Computer Science, 2: 1-18, 1997.

[17] E. de Klerk, D.V. Pasechnik and J.P. Warners. On approximate graph colouring and MAX k-CUT algorithms based on the $\theta$-function. Journal of Combinatorial Optimization, 8(3): 267-294, 2004.

[18] J. Liu, Y. Peng, C. Zhao. Generalized $k$-multiway cut problems. Journal of Applied Mathematics and Computing, 21: 69-82, 2006.

[19] J. B. Orlin, A. P. Punnen, A. S. Schulz. Approximate local search in combinatorial optimization. SIAM Journal on Computing, 33(5): 1201-1214, 2004.

[20] C. H. Papadimitriou. Computational Complexity. Addison-Wesley, 1994.

[21] P. Raghavendra and N. Tan. Approximating CSPs with global cardinality constraints using SDP hierarchies. In SODA'12, pages 373-387, 2012. Full version available as arXiv eprint 1110.1064.

[22] A. A. Schaffer and M. Yannakakis. Simple local search problems that are hard to solve. SIAM Journal on Computing, 20(1): 56-87, 1991.

[23] V. Vazirani. Approximation Algorithms. Springer, 2001.

[24] J. Wu, W. Zhu. On a local search algorithm for the capacitated max-k-cut problem. Kuwait Journal of Science, 41(3): 129-137, 2014.

[25] Y. Ye. A 699-approximation algorithm for Max-Bisection. Mathematical Programming, 90(1): 101-111, 2001.

[26] W. Zhu, C. Guo. A local search approximation algorithm for max-k-cut of graph and hypergraph, 2011 Fourth International Symposium on Parallel Architectures Algorithms and Programming (PAAP), pages 236-240, 2011.

# Chapter 5

# Conclusions

Despite the conceptual simplicity of local search algorithms and the fact that they are rather natural approaches for the solution of optimization problem, local search has not been extensively used in the design of approximation algorithms. The goal of this research is to study the effectiveness of local search in the design of efficient approximation algorithms for NP-hard combinatorial optimization problems. In this chapter first we summarize the main contributions of this research, then we discuss the challenges of using local search in the design of approximation algorithms, and finally we talk about the strengths of local search in the computation of approximate solutions for NP-hard problems.

## 5.1 Main Contributions

We have designed local search approximation algorithms for three well-known NP-hard clustering optimization problems: $k$-facility location, multiway cut, and max $k$-cut.

We have shown how to compute the locality gap for all these algorithms through the use of the local optimality property, namely that the cost of a local optimal solution is better than the cost of all its neighboring solutions, even those containing parts of a global optimum solution. We show how to use the local optimality property to determine a set of inequalities that when carefully combined can be used to bound the cost of a global optimal solution in terms of the cost of a local optimal solution.

We believe that a similar procedure as that used to compute the locality gap of our algorithms could be used in the design and analysis of local search approximation algorithms for other problems.

The locality gap of our algorithm for the $k$-facility location problem matches that of the best known algorithm for the problem, proposed by Zhang [9]. We also obtained a second bound on the locality gap of the algorithm that is better than that of Zhang's algorithm in many cases.

Our local search algorithm for the multiway cut problem has locality gap $2 - \frac{2}{k}$, the same as the approximation ratio of the isolation heuristic by Dahlhaus et al. [4]. However, we have shown that in practice our local search algorithm outperforms the isolation heuristic and has comparable performance to the three currently best known

approximation algorithms for the multiway cut problem: the algorithm of Calinescu et al. [3], the algorithm of Sharma and Vondrak [6], and the algorithm of Buchbinder et al. [1, 2].

Our local search algorithm for the constrained max $k$-cut problem on hypergraphs has locality gap $1 - \frac{1}{k}$, which matches the best known approximation ratio by Vazirani [7] for the max $k$-cut problem on graphs. The constrained max $k$-cut problem generalized problems such as the max Steiner $k$-cut problem, the max $k$-cut problem, and the max multiway cut problem. Also, we have shown that with a slight change in our local search algorithm we can obtain a locality gap of $1 - \frac{|V_{max}|}{|V|}$, for the max $k$-cut problem with given sizes of parts and for the capacitated max $k$-cut problem, where $|V_{max}|$ is the cardinality of the biggest partition produced by our algorithm. The best known algorithm for the capacitated max $k$-cut problem on graphs is by Wu and Zhu [8] and it has approximation ratio $\frac{|V_{min}|(k-1)}{2(|V_{max}|-1)+|V_{min}|(k-1)}$, where $|V_{min}|$ and $|V_{max}|$ are the sizes of the smallest and largest partitions returned by the algorithm, respectively. Our algorithm for the capacitated max $k$-cut problem has approximation ratio $1 - \frac{|V_{max}|}{|V|} \geq 1 - \frac{|V_{max}|}{|V_{max}|+|V_{min}|(k-1)} = \frac{|V_{min}|(k-1)}{|V_{max}|+|V_{min}|(k-1)}$. Therefore, our algorithm is better than the algorithm of Wu and Zhu when $|V_{max}|+|V_{min}|(k-1) < 2(|V_{max}|-1) + |V_{min}|(k-1)$, or in other words, when $|V_{max}| \geq 2$. In addition, our algorithm works on hypergraphs so it is more general than the algorithm of Wu and Zhu.

## 5.2 Challenges of Using Local Search in the Design of Approximation Algorithms

As mentioned in the introduction there are two main challenges that arise from the use of local search in designing approximation algorithms: Analysing the quality of the solutions produced by a local search algorithm might be very complicated, and local search algorithm might return solutions of costs that are very far from those of global optimal solutions.

To overcome the first challenge, we designed ways of taking advantage of the local optimality property of local optimum solutions. This is the key property that allowed us to compare the cost of local and global optimum solutions. We use the local optimal property to obtain a carefully selected set of inequalities that relate parts of the cost of a local optimal solution to parts of the cost of a global optimal one so that when we combine the inequalities we can bound the cost of a local optimal solution in terms of the cost of a global optimal one.

To prevent a local search algorithm from getting trapped in a local optimal solution that is far away from a global optimal one we need to select the local operations carefully. As an example, for the $k$-uncapacitated facility location problem we selected the more complex multi-swaps as local operations instead of the simpler single swaps to increase the size of the of neighborhood of each solution and to improve the chances of escaping from bad local optimal solutions. Similarly, for the minimum multiway cut problem we defined a rather complicated relabel operation as the local operation that can simultaneously change the labels of all the nodes in some set $A$ instead of a simpler operation that

would change the label of a single node. Finally, for the max $k$-cut problem to increase the chances of escaping from bad local optimal solutions we defined two different local operations, not just one: Moving one node to other partitions and swapping two nodes in two different partitions.

Another challenge in designing efficient local search approximation algorithms is ensuring that they have low time complexity. As mentioned in Chapter 1 not all local search algorithms have polynomial running time. The running time of a local search algorithm depends on the number of times that it needs to apply the local operations. The technique by Orlin et al. [5] is of great help to bound the number of times that local operations need to be performed by requiring that local each operation performs a minimum amount of improvement on the value of a solution. Careful application of this technique guarantees a polynomial running time with only a minimum penalty in the quality of the solutions provided by a local search algorithm.

## 5.3 Strengths of Local Search Algorithms

**Conceptual Simplicity:** In order to design a local search iterative improvement algorithm for a combinatorial optimization problem we only need to specify the local operations and terminating condition.

**Easy implementation:** A local search algorithm is a simple iterative algorithm that keeps updating the current solution by looking for a better solution in its neighborhood until no further improvement is possible. Therefore, the only challenging part in implementing a local search algorithm is searching the neighborhood of the current solution to find a solution with a better cost. Sometimes we can take advantage of existing algorithms for other problems; as an example in Chapter 3 for finding a minimum cost relabel operation we model the problem as a maximum flow problem.

**Work well in practice:** We have shown in Chapter 3 that an implementation of our local search algorithm for the multiway cut problem has better performance in practice than other more complex algorithms. Since, the local optimal solutions computed by a local search algorithm partially depend on the initial solution, by considering different initial solutions and selecting the best local optimal solution found we increase the chance of obtaining near-optimum solutions. Furthermore, in practice local search algorithms are fast because usually after a few iterations they find local optimal solutions.

# Bibliography


[1] N. Buchbinder, J. Naor, R. Schwartz, Simplex partitioning via exponential clocks and the multiway cut problem, *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, (2013), 535-544.

[2] N. Buchbinder, J. Naor, R. Schwartz, Simplex transformations and the multiway cut problem, *Proceedings of the Twenty-eight Annual ACM-SIAM Symposium on Discrete Algorithms*, (2016).

[3] G. Calinescu, H. Karloff, Y. Rabani, An Improved Approxmation Algorithm for MULTIWAY CUT. *Journal of Computer and System Sciences*, 3, (2000), 564-574.

[4] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, M. Yannakakis, The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23, (1994), 864-894.

[5] J. B. Orlin, A. P. Punnen, A. S. Schulz. Approximate local search in combinatorial optimization. *SIAM Journal on Computing*, 33(5), (2004) 1201-1214.

[6] A. Sharma, J. Vondrak, Multiway cut, pairwise realizable distribution, and descending thresholds. *In Proceeding of the 46th Annual ACM Symposium on Theory of Computing,* ACM, (2014), 724-733.

[7] V. Vazirani. Approximation Algorithms. *Springer*, (2001).

[8] J. Wu, W. Zhu. On a local search algorithm for the capacitated max-k-cut problem. *Kuwait Journal of Science*, 41(3), (2014), 129-137, .

[9] P. Zhang, A new approximation algorithm for the k-facility location problem. *Theoretical Computer Science*, 384, (2007), 126-135.

# Curriculum Vitae

| | |
|---|---|
| **Name:** | Nasim Samei |
| **Education and Degrees:** | Sharif University of Technology<br>2005 - 2009 B.A.<br><br>University of Western Ontario<br>London, ON<br>2011 - 2013 M.S. |
| **Related Work Experience:** | Teaching Assistant<br>The University of Western Ontario<br>2011 - 2018 |